

Universal Serial Bus

Power Delivery Specification

<i>Revision:</i>	3.2
<i>Version:</i>	1.1
<i>Release Date:</i>	2024-10

Revision History

Revision	Version	Comments	Issue Date
1.0	1.0	Initial release Revision 1.0	5 July, 2012
1.0	1.1	Including errata through 31-October-2012	31 October 2012
1.0	1.2	Including errata through 26-June-2013	26 June, 2013
1.0	1.3	Including errata through 11-March-2014	11 March 2014
2.0	1.0	Initial release Revision 2.0	11 August 2014
2.0	1.1	Including errata through 7-May 2015	7 May 2015
2.0	1.2	Including errata through 25-March-2016	25 March 2016
2.0	1.3	Including errata through 11-January-2017	11 January 2017
3.0	1.0	Initial release Revision 3.0	11 December 2015
3.0	1.0a	Including errata through 25-March-2016	25 March 2016
3.0	1.1	Including errata through 12-January-2017	12 January 2017
3.0	1.2	Including errata through 21-June-2018	21 June 2018
3.0	2.0	Including errata through 29-August-2019	29 August 2019
3.1	1.0	Including errata through May 2021	May 2021
3.1	1.1	Including errata through July 2021 This version incorporates the following ECNs: <ul style="list-style-type: none"> • EPR Clarifications • Define AMS starting point 	July 2021
3.1	1.2	Including errata through October 2021 This version incorporates the following ECNs: <ul style="list-style-type: none"> • Clarify use of Retries • Battery Capabilities • FRS timing problem • PPS power rule clarifications • Peak current support for EPR AVS APDO 	October 2021
3.1	1.3	This version incorporates the following ECNs: <ul style="list-style-type: none"> • Robust EPR Source Operation • EPR Source Caps Editorial • SRC PPS behavior in low current request • Enter USB 	January 2022
3.1	1.4	Editorial changes This version incorporates the following ECNs: <ul style="list-style-type: none"> • Capabilities Mismatch Update • Chunking Timing Issue • OT Mitigation 	April 2022

Revision	Version	Comments	Issue Date
3.1	1.5	Editorial changes This version incorporates the following ECNs: <ul style="list-style-type: none"> • Timer Description Corrections • Change Source_Info Requirements • AMS Update 	July 2022
3.1	1.6	Editorial changes This version incorporates the following ECNs: <ul style="list-style-type: none"> • USB4® V2 Updates • Data Reset Issues • Increase tSenderResponse • PPS Power Limit Bit Update • Support for Asymmetric Mode • Timer Description Corrections Revisited 	October 2022
3.1	1.7	Editorial Changes This version incorporates the following ECNs: <ul style="list-style-type: none"> • Data Reset Invalid Reject Handling • Source request • Source Transition • EPR Entry 	January 2023
3.1	1.8	Editorial Changes This version incorporates the following ECNs: <ul style="list-style-type: none"> • Slew rate exemption for Power Role Swap. • EUDO cable speed clarification. • Update to PPS Requirements. • Deprecate Interruptibility. • Section 7.3 restructure and update. 	April 2023
3.1	1.9	Editorial Changes	July 2023

Revision	Version	Comments	Issue Date
3.2	1.0	<p>This version incorporates the following ECNs:</p> <ul style="list-style-type: none"> • VDM-use Conditions. • tTypeCSinkWaitCap. • tFirstSourceCap Clarification • Hard Reset Clarification. • Unrecognized Country Code • EPR Entry Process-1 • SPR AVS Definition • EPR Power Rules Clarifications 	October 2023
3.2	1.1	<p>This version incorporates the following ECNs:</p> <ul style="list-style-type: none"> • Power Transition time from EPR to PR_Swap • Capabilities Mismatch Update • Deprecate GotoMin and GiveBack Features and Update Power Reserve • EPR Entry requirements Clarification • EPRMDO and Entry Clarification. • Remove 10.2.4 power sharing between ports • Source PDP rating field clarifications • Source Power Rules update. • Source_Info Message Clarifications. • Correction to BMC description. • EPR Source cap clarification. • Delaying of VCONN Swap. • EPR_Request in SPR Mode. • Generic transition diagram. • Removing the usage of Ping message • Sink Standby • Source Info Support 	October 2024

LIMITED COPYRIGHT LICENSE

THE USB 3.0 PROMOTERS GRANT A CONDITIONAL COPYRIGHT LICENSE UNDER THE COPYRIGHTS EMBODIED IN THE USB POWER DELIVERY SPECIFICATION TO USE AND REPRODUCE THE SPECIFICATION FOR THE SOLE PURPOSE OF, AND SOLELY TO THE EXTENT NECESSARY FOR, EVALUATING WHETHER TO IMPLEMENT THE SPECIFICATION IN PRODUCTS THAT WOULD COMPLY WITH THE SPECIFICATION. WITHOUT LIMITING THE FOREGOING, USE THE OF SPECIFICATION FOR THE PURPOSE OF FILING OR MODIFYING ANY PATENT APPLICATION TO TARGET THE SPECIFICATION OR USB COMPLIANT PRODUCTS IS NOT AUTHORIZED. EXCEPT FOR THIS EXPRESS COPYRIGHT LICENSE, NO OTHER RIGHTS OR LICENSES ARE GRANTED, INCLUDING WITHOUT LIMITATION ANY PATENT LICENSES. IN ORDER TO OBTAIN ANY ADDITIONALLY INTELLECTUAL PROPERTY LICENSES OR LICENSING COMMITMENTS ASSOCIATED WITH THE SPECIFICATION A PARTY MUST EXECUTE THE USB 3.0 ADOPTERS AGREEMENT. NOTE: BY USING THE SPECIFICATION, YOU ACCEPT THESE LICENSE TERMS ON YOUR OWN BEHALF AND, IN THE CASE WHERE YOU ARE DOING THIS AS AN EMPLOYEE, ON BEHALF OF YOUR EMPLOYER.

INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED TO YOU "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE. THE AUTHORS OF THIS SPECIFICATION DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS, RELATING TO USE OR IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. THE PROVISION OF THIS SPECIFICATION TO YOU DOES NOT PROVIDE YOU WITH ANY LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS.

USB Type-C[®] and USB4[®] are trademarks of the Universal Serial Bus Implementers Forum (USB-IF).

USB Type-C[®], USB-C[®] and USB4[®] are trademarks of the Universal Serial Bus Implementers Forum (USB-IF). DisplayPort[™] is a trademark of VESA. Thunderbolt[™] is a trademark of Intel Corporation. All product names are trademarks, registered trademarks, or service marks of their respective owners.

You may only use the Thunderbolt[™] trademark or logo in conjunction with products designed to this specification that complete proper certification and executing a Thunderbolt[™] trademark license - see <http://usb.org/compliance> for further information.

All product names are trademarks, registered trademarks, or service marks of their respective owners.

Copyright © 2010- 2024, USB 3.0 Promoter Group: Apple Inc., Hewlett-Packard Inc., Intel Corporation, Microsoft Corporation, Renesas, STMicroelectronics, and Texas Instruments.

All rights reserved.

Please send comments via electronic mail to techsup@usb.org.

For industry information, refer to the USB Implementers Forum web page at <http://www.usb.org>.

Editors

Bob Dunstan
Richard Petrie

Contributors

Charles Wang	ACON, Advanced-Connectek, Inc.	Zaki Moussaoui	Apple
Conrad Choy	ACON, Advanced-Connectek, Inc.	Jeff Liu	ASMedia Technology Inc.
Dennis Chuang	ACON, Advanced-Connectek, Inc.	Kuo Lung Li	ASMedia Technology Inc.
Steve Sedio	ACON, Advanced-Connectek, Inc.	Ming-Wei Hsu	ASMedia Technology Inc.
Sunney Yang	ACON, Advanced-Connectek, Inc.	PS Tseng	ASMedia Technology Inc.
Vicky Chuang	ACON, Advanced-Connectek, Inc.	Sam Tzeng	ASMedia Technology Inc.
Joseph Scanlon	Advanced Micro Devices	Thomas Hsu	ASMedia Technology Inc.
Sujan Thomas	Advanced Micro Devices	Weikao Chang	ASMedia Technology Inc.
Caspar Lin	Allion Labs, Inc.	Yang Cheng	ASMedia Technology Inc.
Casper Lee	Allion Labs, Inc.	Aaron Hou	Bizlink Technology Inc.
Danny Shih	Allion Labs, Inc.	Shawn Meng	Bizlink Technology Inc.
Howard Chang	Allion Labs, Inc.	Bernard Shyu	Bizlink Technology, Inc.
Greg Stewart	Analogix Semiconductor, Inc.	Eric Wu	Bizlink Technology, Inc.
Mehran Badii	Analogix Semiconductor, Inc.	Morphy Hsieh	Bizlink Technology, Inc.
Alexei Kosut	Apple	Sean O'Neal	Bizlink Technology, Inc.
Bill Cornelius	Apple	Tiffany Hsiao	Bizlink Technology, Inc.
Carlos Colderon	Apple	Weichung Ooi	Bizlink Technology, Inc.
Chris Uiterwijk	Apple	Rahul Bhushan	Broadcom Corp.
Colin Whitby-Strevens	Apple	Asila nahas	Cadence Design Systems, Inc.
Corey Axelowitz	Apple	Claire Ying	Cadence Design Systems, Inc.
Corey Lange	Apple	Jie min	Cadence Design Systems, Inc.
Dave Conroy	Apple	Mark Summers	Cadence Design Systems, Inc.
David Sekowski	Apple	Michal Staworko	Cadence Design Systems, Inc.
Girault Jones	Apple	Sathish Kumar Ganesan	Cadence Design Systems, Inc.
James Orr	Apple	Alessandro Ingrassia	Canova Tech
Jason Chung	Apple	Andrea Colognese	Canova Tech
Jay Kim	Apple	Antonio Orzelli	Canova Tech
Jeff Wilcox	Apple	Davide Ghedin	Canova Tech
Jennifer Tsai	Apple	Matteo Casalin	Canova Tech
Karl Bowers	Apple	Michael Marioli	Canova Tech
Keith Porthouse	Apple	Nicola Scantamburlo	Canova Tech
Kevin Hsiue	Apple	Paolo Pilla	Canova Tech
Matt Mora	Apple	Ray Huang	Canyon Semiconductor
Paul Baker	Apple	Yi-Feng Lin	Canyon Semiconductor
Reese Schreiber	Apple	YuHung Lin	Canyon Semiconductor
Ricardo Janezic Pregitzer	Apple	David Tsai	Chrontel, Inc.
Ruchi Chaturvedi	Apple	Anshul Gulati	Cypress Semiconductor
Sameer Kelkar	Apple	Anup Nayak	Cypress Semiconductor
Sasha Tietz	Apple	Benjamin Kropf	Cypress Semiconductor
Scott Jackson	Apple	Dhanraj Rajput	Cypress Semiconductor
Sree Raman	Apple	Ganesh Subramaniam	Cypress Semiconductor
William Ferry	Apple	Jagadeesan Raj	Cypress Semiconductor

Junjie cui	Cypress Semiconductor	Chien-Cheng Kuo	eEver Technology, Inc.
Manu Kumar	Cypress Semiconductor	Shyanjia Chen	eEver Technology, Inc.
Muthu M	Cypress Semiconductor	Abel Astley	Ellisys
Nicholas Bodnaruk	Cypress Semiconductor	Chuck Trefts	Ellisys
Pradeep Bajpai	Cypress Semiconductor	Emmanuel Durin	Ellisys
Rajaram R	Cypress Semiconductor	Mario Pasquali	Ellisys
Rama Vakkantula	Cypress Semiconductor	Tim Wei	Ellisys
Rushil Kadakia	Cypress Semiconductor	Chien-Cheng Kuo	Etron Technology, Inc.
Simon Nguyen	Cypress Semiconductor	Jack Yang	Etron Technology, Inc.
Steven Wong	Cypress Semiconductor	Richard Crisp	Etron Technology, Inc.
Subu Sankaran	Cypress Semiconductor	Shyanjia Chen	Etron Technology, Inc.
Sumeet Gupta	Cypress Semiconductor	TsungTa Lu	Etron Technology, Inc.
Tejender Sheoran	Cypress Semiconductor	Christian Klein	Fairchild Semiconductor
Venkat Mandagulathar	Cypress Semiconductor	Oscar Freitas	Fairchild Semiconductor
Xiaofeng Shen	Cypress Semiconductor	Souhib Harb	Fairchild Semiconductor
Zeng Wei	Cypress Semiconductor	Amanda Ying	Feature Integration Technology Inc.
Adie Tan	Dell Inc.	Jacky Chan	Feature Integration Technology Inc.
Adolfo Montero	Dell Inc.	Kenny Hsieh	Feature Integration Technology Inc.
Bruce Montag	Dell Inc.	KungAn Lin	Feature Integration Technology Inc.
Gary Verdun	Dell Inc.	Paul Yang	Feature Integration Technology Inc.
Ken Nicholas	Dell Inc.	Su Jaden	Feature Integration Technology Inc.
Marcin Nowak	Dell Inc.	Yu-Lin Chu	Feature Integration Technology Inc.
Merle Wood	Dell Inc.	Yulin Lan	Feature Integration Technology Inc.
Mohammed Hijazi	Dell Inc.	AJ Yang	Foxconn / Hon Hai
Siddhartha Reddy	Dell Inc.	Bob Hall	Foxconn / Hon Hai
Terry Matula	Dell Inc.	Chihyin Kan	Foxconn / Hon Hai
Jay Hu	Derun Semiconductor	Fred Fons	Foxconn / Hon Hai
Shelly Liu	Derun Semiconductor	Jie Zheng	Foxconn / Hon Hai
Bindhu Vasu	Dialog Semiconductor (UK) Ltd	Patrick Casher	Foxconn / Hon Hai
Chanchal Gupta	Dialog Semiconductor (UK) Ltd	Shruti Deore	Foxconn / Hon Hai
Dipti Baheti	Dialog Semiconductor (UK) Ltd	Steve Sedio	Foxconn / Hon Hai
Duc Doan	Dialog Semiconductor (UK) Ltd	Terry Little	Foxconn / Hon Hai
Holger Petersen	Dialog Semiconductor (UK) Ltd	Bob McVay	Fresco Logic Inc.
Jianming Yao	Dialog Semiconductor (UK) Ltd	Christopher Meyers	Fresco Logic Inc.
John Shi	Dialog Semiconductor (UK) Ltd	Dian Kurniawan	Fresco Logic Inc.
KE Hong	Dialog Semiconductor (UK) Ltd	Tom Burton	Fresco Logic Inc.
Kevin Mori	Dialog Semiconductor (UK) Ltd	Abraham Levkoy	Google Inc.
Larry Ping	Dialog Semiconductor (UK) Ltd	Adam Rodriguez	Google Inc.
Mengfei Liu	Dialog Semiconductor (UK) Ltd	Alec Berg	Google Inc.
Scott Brown	Dialog Semiconductor (UK) Ltd	Bartosz Szpila	Google Inc.
Yimin Chen	Dialog Semiconductor (UK) Ltd	Benson Leung	Google Inc.
Yong Li	Dialog Semiconductor (UK) Ltd	Chao Fei	Google Inc.
Justin Lee	Diodes Incorporated	Dave Bernard	Google Inc.
Dan Ellis	DisplayLink (UK) Ltd.	David Schneider	Google Inc.
Jason Young	DisplayLink (UK) Ltd.	Diana Zigterman	Google Inc.
Kevin Jacobs	DisplayLink (UK) Ltd.	Eric Herrmann	Google Inc.
Paulo Alcobia	DisplayLink (UK) Ltd.	George-Daniel Matei	Google Inc.
Peter Burgers	DisplayLink (UK) Ltd.	Jameson Thies	Google Inc.
Richard Petrie	DisplayLink (UK) Ltd.	Jim Guerin	Google Inc.

Juan Fantin	Google Inc.	Lee Leppo	HP Inc.
Ken Wu	Google Inc.	Rahul Lakdawala	HP Inc.
Kyle Tso	Google Inc.	Robin Castell	HP Inc.
Mark Hayter	Google Inc.	Roger Benson	HP Inc.
Nathan Kolluru	Google Inc.	Steve Chen	HP Inc.
Nithya Jagannathan	Google Inc.	Bai Sean	Huawei Technologies Co., Ltd.
Srikanth Lakshmikanthan	Google Inc.	Chunjiang Zhao	Huawei Technologies Co., Ltd.
Todd Broch	Google Inc.	JianQuan Wu	Huawei Technologies Co., Ltd.
Toshak Singhal	Google Inc.	Li Zongjian	Huawei Technologies Co., Ltd.
Vincent Palatin	Google Inc.	Liansheng Zheng	Huawei Technologies Co., Ltd.
Xuelin Wu	Google Inc.	Lihua Duan	Huawei Technologies Co., Ltd.
Zhenxue Xu	Google Inc.	Min Chen	Huawei Technologies Co., Ltd.
Alan Kinningham	Granite River Labs	Wang Feng	Huawei Technologies Co., Ltd.
Anand Murugan	Granite River Labs	Wei Haihong	Huawei Technologies Co., Ltd.
Balamurugan Manialagan	Granite River Labs	Zhenning Shi	Huawei Technologies Co., Ltd.
Medipalli Sowmya	Granite River Labs	James Xie	Hynetek Semiconductor Co., Ltd
Mike Engbretson	Granite River Labs	Yingyang Ou	Hynetek Semiconductor Co., Ltd
Mike Wu	Granite River Labs	Robert Heaton	Indie Semiconductor
Mukesh Tatiya	Granite River Labs	Vincent Wang	Indie Semiconductor
Naresh Botsa	Granite River Labs	Benjamin Kropf	Infineon Technologies
PoornaKumar M.	Granite River Labs	Sie Boo Chiang	Infineon Technologies
Prajwal Rathod	Granite River Labs	Tue Fatt David Wee	Infineon Technologies
Rajaraman V	Granite River Labs	Wee Tar Richard Ng	Infineon Technologies
Saai Goutham Revathi Selvam	Granite River Labs	Wolfgang Furtner	Infineon Technologies
Sivan Perumal	Granite River Labs	Aruni Nelson	Intel Corporation
Sivaram Murugesan	Granite River Labs	Bob Dunstan	Intel Corporation
Tim Lin	Granite River Labs	Brad Saunders	Intel Corporation
Vijay S.	Granite River Labs	Chee Lim Nge	Intel Corporation
Vijayakumar P	Granite River Labs	Christine Krause	Intel Corporation
Vishal Kakade	Granite River Labs	Chuen Ming Tan	Intel Corporation
Yogeshwaran Venkatesan	Granite River Labs	Dan Froelich	Intel Corporation
Jerry Qin	GuangDong OPPO Mobile Telecommunications Corp., Ltd.	David Harriman	Intel Corporation
Alan Berkema	Hewlett Packard	David Hines	Intel Corporation
Lee Atkinson	Hewlett Packard	David Thompson	Intel Corporation
Rahul Lakdawala	Hewlett Packard	Guobin Liu	Intel Corporation
Robin Castell	Hewlett Packard	Harry Skinner	Intel Corporation
Ron Schooley	Hewlett Packard	Henrik Leegaard	Intel Corporation
Steve Chen	Hewlett Packard	Jenn Chuan Cheng	Intel Corporation
Suketa Partiwalla	Hewlett Packard	Jervis Lin	Intel Corporation
Vaibhav Malik	Hewlett Packard	John Howard	Intel Corporation
Walter Fry	Hewlett Packard	Karthi Vadivelu	Intel Corporation
Hideyuki HAYAFUJI	Hosiden Corporation	Leo Heiland	Intel Corporation
Keiji Mine	Hosiden Corporation	Maarit Harkonen	Intel Corporation
Masaki Yamaoka	Hosiden Corporation	Nge Chee Lim	Intel Corporation
Takashi Muto	Hosiden Corporation	Paul Durley	Intel Corporation
Yasunori Nishikawa	Hosiden Corporation	Rahman Ismail	Intel Corporation
Alan Berkema	HP Inc.	Rajaram Regupathy	Intel Corporation
Kenneth Chan	HP Inc.	Ronald Swartz	Intel Corporation
Lee Atkinson	HP Inc.	Sarah Sharp	Intel Corporation

Steve McGowan	Intel Corporation	Eric Wen	Luxshare-ICT
Tim McKee	Intel Corporation	James Kirk	Luxshare-ICT
Toby Opferman	Intel Corporation	James Stevens	Luxshare-ICT
Uma Medepalli	Intel Corporation	Josue Castillo	Luxshare-ICT
Venkataramani Gopalakrishnan	Intel Corporation	Pat Young	Luxshare-ICT
Ziv Kabiry	Intel Corporation	Scott Shuey	Luxshare-ICT
Jia Wei	Intersil Corporation	Stone Lin	Luxshare-ICT
Weijie Huang	iST - Integrated Service Technology Inc.	Chikara Kakizawa	Maxim Integrated Products
Al Hsiao	ITE Tech. Inc.	Jacob Scott	Maxim Integrated Products
Greg Song	ITE Tech. Inc.	Ken Helfrich	Maxim Integrated Products
Richard Guo	ITE Tech. Inc.	Michael Miskho	Maxim Integrated Products
Victor Lin	ITE Tech. Inc.	Chris Yokum	MCCI Corporation
Y.C. Chou	ITE Tech. Inc.	Geert Knapen	MCCI Corporation
Kenta Minejima	Japan Aviation Electronics Industry Ltd. (JAE)	Terry Moore	MCCI Corporation
Mark Saubert	Japan Aviation Electronics Industry Ltd. (JAE)	Velmurugan Selvaraj	MCCI Corporation
Toshio Shimoyama	Japan Aviation Electronics Industry Ltd. (JAE)	Tung-Sheng Lin	MediaTek Inc.
Brian Fetz	Keysight Technologies Inc.	Satoru Kumashiro	MegaChips Corporation
Jit Lim	Keysight Technologies Inc.	Brian Marley	Microchip Technology Inc.
Koji Asakawa	Kinetic Technologies Inc.	Dave Perchlik	Microchip Technology Inc.
Babu Mailachalam	Lattice Semiconductor Corp	Don Perkins	Microchip Technology Inc.
Gianluca Mariani	Lattice Semiconductor Corp	Fernando Gonzalez	Microchip Technology Inc.
Joel Coplen	Lattice Semiconductor Corp	John Sisto	Microchip Technology Inc.
Thomas Watza	Lattice Semiconductor Corp	Josh Averyt	Microchip Technology Inc.
Vesa Lauri	Lattice Semiconductor Corp	Kiet Tran	Microchip Technology Inc.
Bruce Chuang	Leadtrend	Mark Bohm	Microchip Technology Inc.
Eilian Liu	Leadtrend	Matthew Kalibat	Microchip Technology Inc.
Chetan Kopalle	LeCroy Corporation	Mick Davis	Microchip Technology Inc.
Daniel H Jacobs	LeCroy Corporation	Prasanna Vengateshan	Microchip Technology Inc.
Jake Jacobs	LeCroy Corporation	Rich Wahler	Microchip Technology Inc.
Kimberley McKay	LeCroy Corporation	Richard Petrie	Microchip Technology Inc.
Mike Engbretson	LeCroy Corporation	Ronald Kunin	Microchip Technology Inc.
Mike Micheletti	LeCroy Corporation	Shannon Cash	Microchip Technology Inc.
Roy Chestnut	LeCroy Corporation	Thomas Farkas	Microchip Technology Inc.
Tyler Joe	LeCroy Corporation	Venkataraman Krishnamoorthy	Microchip Technology Inc.
Phil Jakes	Lenovo	Andrew Yang	Microsoft Corporation
Do Kyun Kim	LG electronics	Anthony Chen	Microsoft Corporation
Won-Jong Choi	LG electronics	Arvind Murching	Microsoft Corporation
Won-Jong Choi	LG Electronics Ltd.	Dave Perchlik	Microsoft Corporation
Aaron Melgar	Lion Semiconductor	David Voth	Microsoft Corporation
Chris Zhou	Lion Semiconductor	Geoff Shew	Microsoft Corporation
Sehyung Jeon	Lion Semiconductor	Jayson Kastens	Microsoft Corporation
Wonyoung Kim	Lion Semiconductor	Kai Inha	Microsoft Corporation
Yongho Kim	Lion Semiconductor	Marwan Kadado	Microsoft Corporation
Dave Thompson	LSI Corporation	Michelle Bergeron	Microsoft Corporation
Alan Kinningham	Luxshare-ICT	Nathan Sherman	Microsoft Corporation
Alan Liu	Luxshare-ICT	Rahul Ramadas	Microsoft Corporation
Scott Brenden	Intel Corporation	Randy Aull	Microsoft Corporation
Sridharan Ranganathan	Intel Corporation	Shiu Ng	Microsoft Corporation
Daniel Chen	Luxshare-ICT	Tieyong Yin	Microsoft Corporation

Timo Toivola	Microsoft Corporation	Hung-Chih Chiu	Power Forest Technology Corporation
Toby Nixon	Microsoft Corporation	Jay Tu	Power Forest Technology Corporation
Vahid Vassey	Microsoft Corporation	Adel Lahham	Power Integrations
Vivek Gupta	Microsoft Corporation	Aditya Kulkarni	Power Integrations
Yang You	Microsoft Corporation	Akshay Nayaknur	Power Integrations
Adib Al Abaji	Molex LLC	Amruta Patra	Power Integrations
Aaron Xu	Monolithic Power Systems Inc.	K R Rahul Raj	Power Integrations
Bo Zhou	Monolithic Power Systems Inc.	Kaushik Raam	Power Integrations
Christian Sporck	Monolithic Power Systems Inc.	Rahul Joshi	Power Integrations
Di Han	Monolithic Power Systems Inc.	Ricardo Pregiteer	Power Integrations
Zhihong Yu	Monolithic Power Systems Inc.	Shruti Anand	Power Integrations
Dan Wagner	Motorola Mobility Inc.	Amit gupta	Qualcomm, Inc
Ben Crowe	MQP Electronics Ltd.	George Paparrizos	Qualcomm, Inc
Pat Crowe	MQP Electronics Ltd.	Giovanni Garcea	Qualcomm, Inc
Sten Carlsen	MQP Electronics Ltd.	Jack Pham	Qualcomm, Inc
Kenji Oguma	NEC Corporation	James Goel	Qualcomm, Inc
ChinJui Lin	Nexperia B.V.	Joshua Warner	Qualcomm, Inc
Max Guan	Nexperia B.V.	Karyn Vuong	Qualcomm, Inc
Stefan Seider	Nexperia B.V.	Lalan Mishra	Qualcomm, Inc
Frank Borngreber	Nokia Corporation	Nicholas Cadieux	Qualcomm, Inc
Kai Inha	Nokia Corporation	Vamsi Samavedam	Qualcomm, Inc
Pekka Leinonen	Nokia Corporation	Vatsal Patel	Qualcomm, Inc
Richard Petrie	Nokia Corporation	Chris Sporck	Qualcomm, Inc.
Sten Carlsen	Nokia Corporation	Craig Aiken	Qualcomm, Inc.
Abhijeet Kulkarni	NXP Semiconductors	Narendra Mehta	Qualcomm, Inc.
Ahmad Yazdi	NXP Semiconductors	Terry Remple	Qualcomm, Inc.
Bart Vertenten	NXP Semiconductors	Will Kun	Qualcomm, Inc.
Dennis Ha	NXP Semiconductors	Yoram Rimoni	Qualcomm, Inc.
Dong Nguyen	NXP Semiconductors	Fan-Hau Hsu	Realtek Semiconductor Corp.
Guru Prasad	NXP Semiconductors	Tsung-Peng Chuang	Realtek Semiconductor Corp.
Ken Jaramillo	NXP Semiconductors	Atsushi Mitamura	Renesas Electronics Corp.
Krishnan TN	NXP Semiconductors	Bob Dunstan	Renesas Electronics Corp.
Michael Joehren	NXP Semiconductors	Brian Allen	Renesas Electronics Corp.
Robert de Nie	NXP Semiconductors	Dan Aoki	Renesas Electronics Corp.
Rod Whitby	NXP Semiconductors	Fengshuan Zhou	Renesas Electronics Corp.
Vijendra Kuroodi	NXP Semiconductors	Hajime Nozaki	Renesas Electronics Corp.
Winston Langeslag	NXP Semiconductors	John Carpenter	Renesas Electronics Corp.
Robert Heaton	Obsidian Technology	Kiichi Muto	Renesas Electronics Corp.
Andrew Yoo	ON Semiconductor	Masami Katagiri	Renesas Electronics Corp.
Brady Maasen	ON Semiconductor	Nobuo Furuya	Renesas Electronics Corp.
Bryan McCoy	ON Semiconductor	Patrick Yu	Renesas Electronics Corp.
Christian Klein	ON Semiconductor	Peter Teng	Renesas Electronics Corp.
Cor Voorwinden	ON Semiconductor	Philip Leung	Renesas Electronics Corp.
Edward Berrios	ON Semiconductor	Steve Roux	Renesas Electronics Corp.
Michael Smith	ON Semiconductor	Tetsu Sato	Renesas Electronics Corp.
Oscar Freitas	ON Semiconductor	Toshifumi Yamaoka	Renesas Electronics Corp.
Tom Duffy	ON Semiconductor	Yimin Chen	Renesas Electronics Corp.
Brian Collins	Parade Technologies Inc.	Chunan Kuo	Richtek Technology Corporation
Craig Wiley	Parade Technologies Inc.	Heinz Wei	Richtek Technology Corporation

Max Huang	Richtek Technology Corporation	Shannon Cash	SMSC
TZUHSIEN CHUANG	Richtek Technology Corporation	Mark Bohm	SMSC
Tatsuya Irisawa	Ricoh Company Ltd.	Tim Knowlton	SMSC
Akihiro Ono	Rohm Co. Ltd.	William Chiechi	SMSC
Chris Lin	Rohm Co. Ltd.	Shigenori Tagami	Sony Corporation
Hidenori Nishimoto	Rohm Co. Ltd.	Shinichi Hirata	Sony Corporation
Kris Bahar	Rohm Co. Ltd.	Amanda Hosler	Specwerkz
Manabu Miyata	Rohm Co. Ltd.	Bob Dunstan	Specwerkz
Ruben Balbuena	Rohm Co. Ltd.	Brad Saunders	Specwerkz
Takashi Sato	Rohm Co. Ltd.	Diane Lenox	Specwerkz
Vijendra Kuroodi	Rohm Co. Ltd.	Michael Munn	StarTech.com Ltd.
Yusuke Kondo	Rohm Co. Ltd.	Fabien Friess	ST-Ericsson
Kazuomi Nagai	ROHM Co., Ltd.	Giuseppe Platania	ST-Ericsson
Matti Kulmala	Salcomp Plc	Jean-Francois Gatto	ST-Ericsson
Toni Lehimo	Salcomp Plc	Milan Stamenkovic	ST-Ericsson
Edward Lee	Samsung Electronics Co. Ltd.	Nicolas Florenchie	ST-Ericsson
Tong Kim	Samsung Electronics Co. Ltd.	Patrizia Milazzo	ST-Ericsson
Amit Bouzaglo	Scosche Industries	Christophe Cochard	STMicroelectronics
Alvin Cox	Seagate Technology LLC	Christophe Lorin	STMicroelectronics
Emmanuel Lemay	Seagate Technology LLC	Filippo Bonaccorso	STMicroelectronics
John Hein	Seagate Technology LLC	Jessy Guilbot	STMicroelectronics
Marc Noblitt	Seagate Technology LLC	Joel Huloux	STMicroelectronics
Michael Morgan	Seagate Technology LLC	John Bloomfield	STMicroelectronics
Ronald Rueckert	Seagate Technology LLC	Massimo Panzica	STMicroelectronics
Tony Priborsky	Seagate Technology LLC	Meriem Mersel	STMicroelectronics
Chin Chang	Semtech Corporation	Nathalie Ballot	STMicroelectronics
Tom Farkas	Semtech Corporation	Pascal Legrand	STMicroelectronics
Ankit Garg	Siemens Industry Software Inc.	Patrizia Milazzo	STMicroelectronics
Ning Dai	Silergy Corp.	Richard O'Connor	STMicroelectronics
Wanfeng Zhang	Silergy Corp.	Morten Christiansen	Synopsys, Inc.
Kafai Leung	Silicon Laboratories, Inc.	Nivin George	Synopsys, Inc.
Kok Hong Soh	Silicon Laboratories, Inc.	Prishkit Abrol	Synopsys, Inc.
Sorin Badiu	Silicon Laboratories, Inc.	Zongyao Wen	Synopsys, Inc.
Steven Ghang	Silicon Laboratories, Inc.	Joan Marrinan	Tektronix
Abhishek Sardeshpande	SiliConch Systems Private Limited	Kimberley McKay	Teledyne-LeCroy
Aniket Mathad	SiliConch Systems Private Limited	Matthew Dunn	Teledyne-LeCroy
Chandana N	SiliConch Systems Private Limited	Tony Minchell	Teledyne-LeCroy
Jaswanth Ammineni	SiliConch Systems Private Limited	Anand Dabak	Texas Instruments
Jinisha Patel	SiliConch Systems Private Limited	Annamalai Kasthuri	Texas Instruments
Kaustubh Kumar	SiliConch Systems Private Limited	BIJU Erayamkot	Texas Instruments
Nitish	SiliConch Systems Private Limited	Panayamthatta	Texas Instruments
Pavitra Balasubramanian	SiliConch Systems Private Limited	Bill Waters	Texas Instruments
Rakesh Polasa	SiliConch Systems Private Limited	Bing Lu	Texas Instruments
Satish Anand Verkila	SiliConch Systems Private Limited	Deric Waters	Texas Instruments
Shubham Paliwal	SiliConch Systems Private Limited	Grant Ley	Texas Instruments
Vishnu Pusuluri	SiliConch Systems Private Limited	Gregory Watkins	Texas Instruments
John Sisto	SMSC	Ingolf Frank	Texas Instruments
Ken Gay	SMSC	Ivo Huber	Texas Instruments
Richard Wahler	SMSC	Javed Ahmad	Texas Instruments
		Jean Picard	Texas Instruments

John Perry	Texas Instruments
Kasthuri Annamalai	Texas Instruments
Martin Patoka	Texas Instruments
Mike Campbell	Texas Instruments
Scott Jackson	Texas Instruments
Shafiuddin Mohammed	Texas Instruments
Srinath Hosur	Texas Instruments
Steven Tom	Texas Instruments
Yoon Lee	Texas Instruments
Tim Wilhelm	The Silanna Group Pty. Ltd.
Tod Wolf	The Silanna Group Pty. Ltd.
Chris Yokum	Total Phase
Dylan Su	UL LLC
Eric Wall	UL LLC
Jason Smith	UL LLC
Terry Kao	UL LLC
Steven Chen	Unigraf OY
Topi Lampiranta	Unigraf OY
Brad Cox	Ventev Mobile
Colin Vose	Ventev Mobile
Dydron Lin	VIA Technologies, Inc.
Fong-Jim Wang	VIA Technologies, Inc.
Jay Tseng	VIA Technologies, Inc.
Rex Chang	VIA Technologies, Inc.
Terrance Shih	VIA Technologies, Inc.
Ho Wen Tsai	Weltrend Semiconductor
Hung Chiang	Weltrend Semiconductor
Jeng Cheng Liu	Weltrend Semiconductor
Priscilla Lee	Weltrend Semiconductor
Wayne Lo	Weltrend Semiconductor
Charles Neumann	Western Digital Technologies, Inc.
Curtis Stevens	Western Digital Technologies, Inc.
John Maroney	Western Digital Technologies, Inc.
Joe O'Brien	Wilder Technologies
Will Miller	Wilder Technologies
Canfeng Chen	Xiaomi Communications Co., Ltd.
Juejia Zhou	Xiaomi Communications Co., Ltd.
Xiaoxing Yang	Xiaomi Communications Co., Ltd.
Liu Qiong	Zhuhai Smartware Technology Co., Ltd.
Long Zhang	Zhuhai Smartware Technology Co., Ltd.
Yuanchao Liang	Zhuhai Smartware Technology Co., Ltd.

Table Of Contents

Revision History	2
LIMITED COPYRIGHT LICENSE	5
INTELLECTUAL PROPERTY DISCLAIMER	5
Editors	6
Contributors	6
Table Of Contents	13
List of Figures	19
List of Tables	27
1 Introduction	34
1.1 Overview.....	34
1.2 Purpose	35
1.2.1 Scope	36
1.3 Section Overview	36
1.4 Conventions.....	37
1.4.1 Precedence	37
1.4.2 Keywords	37
1.4.3 Numbering	38
1.5 Related Documents	39
1.6 Terms and Abbreviations.....	40
1.7 Parameter Values.....	52
1.8 Changes from Revision 3.0.....	52
1.9 Compatibility with Revision 2.0	52
2 Overview	53
2.1 Introduction	53
2.1.1 Power Delivery Source Operational Contracts	53
2.1.2 Power Delivery Contract Negotiation	53
2.1.3 Other Uses for Power Delivery	54
2.2 Compatibility with Revision 2.0	54
2.3 USB Power Delivery Capable Devices.....	55
2.4 SOP* Communication	57
2.4.1 Introduction	57
2.4.2 SOP* Collision Avoidance	57
2.4.3 SOP Communication	57
2.4.4 SOP'/SOP'' Communication with Cable Plugs	57
2.5 Operational Overview	59
2.5.1 Source Operation	59
2.5.2 Sink Operation	62
2.5.3 Cable Plugs	65
2.6 Architectural Overview	66
2.6.1 Policy	69
2.6.2 Message Formation and Transmission	70
2.6.3 Collision Avoidance	70
2.6.4 Power supply	71
2.6.5 DFP/UFP	71

2.6.6	Cable and Connectors	72
2.6.7	Interactions between Non-PD, BC, and PD devices	72
2.6.8	Power Rules	72
2.7	Extended Power Range (EPR) Operation	73
2.8	Charging Models	75
2.8.1	Fixed Supply Charging Models	75
2.8.2	Programmable Power Supply (PPS) Charging Models	75
2.8.3	Adjustable Voltage Supply (AVS) Charging Models	76
3	USB Type-A and USB Type-B Cable Assemblies and Connectors	77
4	Electrical Requirements	78
4.1	Interoperability with other USB Specifications	78
4.2	Dead Battery Detection / Unpowered Port Detection	78
4.3	Cable IR Ground Drop (IR Drop)	78
4.4	Cable Type Detection	78
5	Physical Layer	79
5.1	Physical Layer Overview	79
5.2	Physical Layer Functions	79
5.3	Symbol Encoding	80
5.4	Ordered Sets	81
5.5	Transmitted Bit Ordering	83
5.6	Packet Format	84
5.6.1	Packet Framing	84
5.6.2	CRC	86
5.6.3	Packet Detection Errors	89
5.6.4	Hard Reset	89
5.6.5	Cable Reset	90
5.7	Collision Avoidance	91
5.8	Bi-phase Mark Coding (BMC) Signaling Scheme	92
5.8.1	Encoding and signaling	92
5.8.2	Transmit and Receive Masks	96
5.8.3	Transmitter Load Model	103
5.8.4	BMC Common specifications	105
5.8.5	BMC Transmitter Specifications	106
5.8.6	BMC Receiver Specifications	110
5.9	Built in Self-Test (BIST)	113
5.9.1	BIST Carrier Mode	113
5.9.2	BIST Test Data Mode	113
6	Protocol Layer	114
6.1	Overview	114
6.2	Messages	115
6.2.1	Message Construction	115
6.3	Control Message	128
6.3.1	GoodCRC Message	129
6.3.2	GotoMin Message (Deprecated)	129
6.3.3	Accept Message	129
6.3.4	Reject Message	129
6.3.5	Ping Message	130
6.3.6	PS_RDY Message	130
6.3.7	Get_Source_Cap Message	131
6.3.8	Get_Sink_Cap Message	131

6.3.9	DR_Swap Message	131
6.3.10	PR_Swap Message	131
6.3.11	VCONN_Swap Message	132
6.3.12	Wait Message	133
6.3.13	Soft Reset Message	134
6.3.14	Data_Reset Message	135
6.3.15	Data_Reset_Complete Message	135
6.3.16	Not_Supported Message	135
6.3.17	Get_Source_Cap_Extended Message	136
6.3.18	Get_Status Message	136
6.3.19	FR_Swap Message	136
6.3.20	Get_PPS_Status	137
6.3.21	Get_Country_Codes	137
6.3.22	Get_Sink_Cap_Extended Message	137
6.3.23	Get_Source_Info Message	137
6.3.24	Get_Revision Message	137
6.4	Data Message	138
6.4.1	Capabilities Message	139
6.4.2	Request Message	155
6.4.3	BIST Message	159
6.4.4	Vendor Defined Message	162
6.4.5	Battery_Status Message	196
6.4.6	Alert Message	198
6.4.7	Get_Country_Info Message	201
6.4.8	Enter_USB Message	202
6.4.9	EPR_Request Message	205
6.4.10	EPR_Mode Message	206
6.4.11	Source_Info Message	212
6.4.12	Revision Message	214
6.5	Extended Message	215
6.5.1	Source_Capabilities_Extended Message	216
6.5.2	Status Message	221
6.5.3	Get_Battery_Cap Message	227
6.5.4	Get_Battery_Status Message	227
6.5.5	Battery_Capabilities Message	228
6.5.6	Get_Manufacturer_Info Message	230
6.5.7	Manufacturer_Info Message	231
6.5.8	Security Messages	233
6.5.9	Firmware Update Messages	234
6.5.10	PPS_Status Message	235
6.5.11	Country_Codes Message	237
6.5.12	Country_Info Message	238
6.5.13	Sink_Capabilities_Extended Message	239
6.5.14	Extended_Control Message	244
6.5.15	EPR Capabilities Message	245
6.5.16	Vendor_Defined_Extended Message	247
6.6	Timers	249
6.6.1	CRCReceiveTimer	249
6.6.2	SenderResponseTimer	249
6.6.3	Capability Timers	249
6.6.4	Wait Timers and Times	251
6.6.5	Power Supply Timers	251
6.6.6	NoResponseTimer	253
6.6.7	BIST Timers	253
6.6.8	Power Role Swap Timers	254
6.6.9	Soft Reset Timers	254
6.6.10	Data Reset Timers	254
6.6.11	Hard Reset Timers	255

6.6.12	Structured VDM Timers	255
6.6.13	VCONN Timers	257
6.6.14	tCableMessage	257
6.6.15	DiscoverIdentityTimer	257
6.6.16	Collision Avoidance Timers	257
6.6.17	Fast Role Swap Timers	258
6.6.18	Chunking Timers	258
6.6.19	Programmable Power Supply Timers	259
6.6.20	tEnterUSB	260
6.6.21	EPR Timers	260
6.6.22	Time Values and Timers	262
6.7	Counters.....	265
6.7.1	MessageID Counter	265
6.7.2	Retry Counter	265
6.7.3	Hard Reset Counter	266
6.7.4	Capabilities Counter	266
6.7.5	Discover Identity Counter	266
6.7.6	VDMBusyCounter	266
6.7.7	Counter Values and Counters	267
6.8	Reset	268
6.8.1	Soft Reset and Protocol Error	268
6.8.2	Data Reset	270
6.8.3	Hard Reset	270
6.8.4	Cable Reset	271
6.9	Accept, Reject and Wait.....	272
6.10	Collision Avoidance.....	273
6.11	Message Discarding.....	274
6.12	State behavior.....	275
6.12.1	Introduction to state diagrams used in Chapter 6	275
6.12.2	State Operation	276
6.12.3	List of Protocol Layer States	298
6.13	Message Applicability	300
6.13.1	Applicability of Control Messages	301
6.13.2	Applicability of Data Messages	303
6.13.3	Applicability of Extended Messages	305
6.13.4	Applicability of Extended Control Messages	308
6.13.5	Applicability of Structured VDM Commands	309
6.13.6	Applicability of Reset Signaling	310
6.13.7	Applicability of Fast Role Swap Request	310
6.14	Value Parameters.....	311
7	Power Supply.....	312
7.1	Source Requirements.....	312
7.1.1	Behavioral Aspects	312
7.1.2	Source Bulk Capacitance	312
7.1.3	Types of Sources	312
7.1.4	Source Transitions	313
7.1.5	Response to Hard Resets	322
7.1.6	Changing the Output Power Capability	323
7.1.7	Robust Source Operation	323
7.1.8	Output Voltage Tolerance and Range	325
7.1.9	Charging and Discharging the Bulk Capacitance on VBUS	328
7.1.10	Swap Standby for Sources	328
7.1.11	Source Peak Current Operation	328
7.1.12	Source Capabilities Extended Parameters	329
7.1.13	Fast Role Swap	331

7.1.14	Non-application of VBUS Slew Rate Limits	332
7.1.15	VCONN Power Cycle	333
7.2	Sink Requirements	335
7.2.1	Behavioral Aspects	335
7.2.2	Sink Bulk Capacitance	335
7.2.3	Sink Standby	335
7.2.4	Suspend Power Consumption	336
7.2.5	Zero Negotiated Current	336
7.2.6	Transient Load Behavior	336
7.2.7	Swap Standby for Sinks	336
7.2.8	Sink Peak Current Operation	336
7.2.9	Robust Sink Operation	337
7.2.10	Fast Role Swap	338
7.3	Transitions.....	339
7.3.1	Transitions caused by a Request Message	340
7.3.2	Transitions Caused by Power Role Swap	388
7.3.3	Transitions Caused by Hard Reset	396
7.3.4	Transitions Caused by Fast Role Swap	400
7.4	Electrical Parameters.....	404
7.4.1	Source Electrical Parameters	404
7.4.2	Sink Electrical Parameters	413
7.4.3	Common Electrical Parameters	415
8	Device Policy	416
8.1	Overview.....	416
8.2	Device Policy Manager.....	417
8.2.1	Capabilities	418
8.2.2	System Policy	418
8.2.3	Control of Source/Sink	419
8.2.4	Cable Detection	419
8.2.5	Managing Power Requirements	420
8.2.6	Use of "Unconstrained Power" bit with Batteries and AC supplies	421
8.2.7	Interface to the Policy Engine	423
8.3	Policy Engine.....	424
8.3.1	Introduction	424
8.3.2	Atomic Message Sequence Diagrams	425
8.3.3	State Diagrams	822
9	States and Status Reporting	975
9.1	Overview.....	975
9.1.1	PDUSB Device and Hub Requirements	979
9.1.2	Mapping to USB Device States	980
9.1.3	PD Software Stack	983
9.1.4	PDUSB Device Enumeration	984
9.2	PD Specific Descriptors	985
9.2.1	USB Power Delivery Capability Descriptor	986
9.2.2	Battery Info Capability Descriptor	987
9.2.3	PD Consumer Port Capability Descriptor	988
9.2.4	PD Provider Port Capability Descriptor	989
9.3	PD Specific Requests and Events.....	990
9.3.1	PD Specific Requests	990
9.4	PDUSB Hub and PDUSB Peripheral Device Requests.....	991
9.4.1	GetBatteryStatus	991
9.4.2	SetPDFeature	992
10	Power Rules	995

10.1	Introduction	995
10.2	Source Power Rules	995
10.2.1	Source Power Rule Considerations	996
10.2.2	Normative Voltages and Currents	997
10.2.3	Optional Voltages/Currents	1005
10.3	Sink Power Rules	1013
10.3.1	Sink Power Rule Considerations	1013
10.3.2	Normative Sink Rules	1013
A	CRC calculation	1015
A.1	C code example	1015
B	Message Sequence Examples (Deprecated)	1016
C	VDM Command Examples	1017
C.1	Discover Identity Example.....	1017
C.1.1	Discover Identity Command request	1017
C.1.2	Discover Identity Command response - Active Cable.	1018
C.1.3	Discover Identity Command response - Hub.	1020
C.2	Discover SVIDs Example	1021
C.2.1	Discover SVIDs Command request	1021
C.2.2	Discover SVIDs Command response	1022
C.3	Discover Modes Example.....	1023
C.3.1	Discover Modes Command request	1023
C.3.2	Discover Modes Command response	1024
C.4	Enter Mode Example	1025
C.4.1	Enter Mode Command request	1025
C.4.2	Enter Mode Command response	1026
C.4.3	Enter Mode Command request with additional VDO.	1027
C.5	Exit Mode Example.....	1028
C.5.1	Exit Mode Command request	1028
C.5.2	Exit Mode Command response	1029
C.6	Attention Example.....	1030
C.6.1	Attention Command request	1030
C.6.2	Attention Command request with additional VDO.	1031
D	BMC Receiver Design Examples	1032
D.1	Finite Difference Scheme	1032
D.1.1	Sample Circuitry	1032
D.1.2	Theory	1032
D.1.3	Data Recovery	1034
D.1.4	Noise Zone and Detection Zone	1035
D.2	Subtraction Scheme	1036
D.2.1	Sample Circuitry	1036
D.2.2	Output of Each Circuit Block	1036
D.2.3	Subtractor Output at Power Source and Power Sink	1036
D.2.4	Noise Zone and Detection Zone	1037
E	FRS System Level Example	1038
E.1	Overview.....	1038
E.2	FRS Initial Setup	1041
E.3	FRS Process	1044

List of Figures

1	Introduction	34
2	Overview	53
Figure 2.1	Logical Structure of USB Power Delivery Capable Devices.....	55
Figure 2.2	Example SOP' Communication between VCONN Source and Cable Plug(s)	58
Figure 2.3	USB Power Delivery Communications Stack.....	67
Figure 2.4	USB Power Delivery Communication Over USB	68
Figure 2.5	High Level Architecture View	69
Figure 2.6	Example of a Normal EPR Mode Operational Flow	74
3	USB Type-A and USB Type-B Cable Assemblies and Connectors	77
4	Electrical Requirements.....	78
5	Physical Layer	79
Figure 5.1	Interpretation of ordered sets	81
Figure 5.2	Transmit Order for Various Sizes of Data.....	83
Figure 5.3	USB Power Delivery Packet Format.....	84
Figure 5.4	CRC-32 Generation	88
Figure 5.5	Line format of Hard Reset.....	89
Figure 5.6	Line format of Cable Reset.....	90
Figure 5.7	BMC Example	92
Figure 5.8	BMC Transmitter Block Diagram.....	92
Figure 5.9	BMC Receiver Block Diagram.....	93
Figure 5.10	BMC Encoded Start of Preamble	93
Figure 5.11	Transmitting or Receiving BMC Encoded Frame Terminated by Zero with High-to-Low Last Transition.....	94
Figure 5.12	Transmitting or Receiving BMC Encoded Frame Terminated by One with High-to-Low Last Transition.....	94
Figure 5.13	Transmitting or Receiving BMC Encoded Frame Terminated by Zero with Low to High Last Transition.....	95
Figure 5.14	Transmitting or Receiving BMC Encoded Frame Terminated by One with Low to High Last Transition	95
Figure 5.15	BMC Tx 'ONE' Mask	96
Figure 5.16	BMC Tx 'ZERO' Mask.....	97
Figure 5.17	BMC Rx 'ONE' Mask when Sourcing Power.....	99
Figure 5.18	BMC Rx 'ZERO' Mask when Sourcing Power	100
Figure 5.19	BMC Rx 'ONE' Mask when Power neutral.....	100
Figure 5.20	BMC Rx 'ZERO' Mask when Power neutral	101
Figure 5.21	BMC Rx 'ONE' Mask when Sinking Power	101
Figure 5.22	BMC Rx 'ZERO' Mask when Sinking Power	102
Figure 5.23	Transmitter Load Model for BMC Tx from a Source	103
Figure 5.24	Transmitter Load Model for BMC Tx from a Sink	103
Figure 5.25	Transmitter diagram illustrating zDriver.....	107
Figure 5.26	Inter-Frame Gap Timings.....	108
Figure 5.27	Example Multi-Drop Configuration showing two DRPs	111
Figure 5.28	Example Multi-Drop Configuration showing a DFP and UFP.....	111
Figure 5.29	Test Frame.....	113
6	Protocol Layer	114
Figure 6.1	USB Power Delivery Packet Format for a Control Message.....	115
Figure 6.2	USB Power Delivery Packet Format including Data Message Payload	116
Figure 6.3	USB Power Delivery Packet Format including an Extended Message Header and Payload	116

Figure 6.4	Example Security_Request sequence Unchunked (Chunked bit = 0)	123
Figure 6.5	Example byte transmission for Security_Request Message of Data Size 7 (Chunked bit is set to zero).....	123
Figure 6.6	Example byte transmission for Security_Response Message of Data Size 30 (Chunked bit is set to zero)	124
Figure 6.7	Example Security_Request sequence Chunked (Chunked bit = 1).....	125
Figure 6.8	Example Security_Request Message of Data Size 7 (Chunked bit set to 1).....	126
Figure 6.9	Example Chunk 0 of Security_Response Message of Data Size 30 (Chunked bit set to 1).....	126
Figure 6.10	Example byte transmission for a Security_Response Message Chunk request (Chunked bit is set to 1).....	127
Figure 6.11	Example Chunk 1 of Security_Response Message of Data Size 30 (Chunked bit set to 1).....	127
Figure 6.12	SPR Capabilities Message Construction.....	152
Figure 6.13	Example Capabilities Message with 2 Power Data Objects.....	152
Figure 6.14	BIST Message	159
Figure 6.15	Vendor Defined Message.....	162
Figure 6.16	Discover Identity Command response.....	169
Figure 6.17	Discover Identity Command response for a DRD.....	169
Figure 6.18	Example Discover SVIDs response with 3 SVIDs.....	186
Figure 6.19	Example Discover SVIDs response with 4 SVIDs.....	187
Figure 6.20	Example Discover SVIDs response with 12 SVIDs followed by an empty response.....	187
Figure 6.21	Example Discover Modes response for a given SVID with 3 Modes	188
Figure 6.22	Successful Enter Mode sequence.....	189
Figure 6.23	Unsuccessful Enter Mode sequence due to NAK.....	190
Figure 6.24	Exit Mode sequence.....	191
Figure 6.25	Attention Command request/response sequence	192
Figure 6.26	Command request/response sequence	193
Figure 6.27	Enter/Exit Mode Process	195
Figure 6.28	Battery_Status Message	196
Figure 6.29	Alert Message.....	198
Figure 6.30	Get_Country_Info Message	201
Figure 6.31	Enter_USB Message	203
Figure 6.32	EPR_Request Message.....	205
Figure 6.33	EPR Mode DO Message.....	206
Figure 6.34	Illustration of process to enter EPR Mode.....	208
Figure 6.35	Source_Info Message.....	212
Figure 6.36	Revision Message Data Object	214
Figure 6.37	Source_Capabilities_Extended Message	216
Figure 6.38	SOP Status Message	221
Figure 6.39	SOP'/SOP'' Status Message.....	226
Figure 6.40	Get_Battery_Cap Message	227
Figure 6.41	Get_Battery_Status Message	227
Figure 6.42	Battery_Capabilities Message.....	228
Figure 6.43	Get_Manufacturer_Info Message.....	230
Figure 6.44	Manufacturer_Info Message.....	231
Figure 6.45	Security_Request Message.....	233
Figure 6.46	Security_Response Message	233
Figure 6.47	Firmware_Update_Request Message	234
Figure 6.48	Firmware_Update_Response Message	234
Figure 6.49	PPS_Status Message.....	235
Figure 6.50	Country_Codes Message	237
Figure 6.51	Country_Info Message	238
Figure 6.52	Sink_Capabilities_Extended Message.....	239
Figure 6.53	Extended_Control Message	244
Figure 6.54	Mapping SPR Capabilities to EPR Capabilities.....	245
Figure 6.55	EPR_Source_Capabilities message with no EPR PDOs	246
Figure 6.56	Vendor_Defined_Extended Message	247
Figure 6.57	Outline of States	275
Figure 6.58	References to states.....	275

Figure 6.59	Chunking architecture Showing Message and Control Flow	277
Figure 6.60	Chunked Rx State Diagram.....	278
Figure 6.61	Chunked Tx State Diagram.....	281
Figure 6.62	Chunked Message Router State Diagram	284
Figure 6.63	Common Protocol Layer Message Transmission State Diagram	286
Figure 6.64	Source Protocol Layer Message Transmission State Diagram.....	289
Figure 6.65	Sink Protocol Layer Message Transmission State Diagram.....	291
Figure 6.66	Protocol layer Message reception	293
Figure 6.67	Hard/Cable Reset	295

7 Power Supply 312

Figure 7.1	Placement of Source Bulk Capacitance	312
Figure 7.2	Transition Envelope for Positive Voltage Transitions	313
Figure 7.3	Transition Envelope for Negative Voltage Transitions.....	314
Figure 7.4	PPS Positive Voltage Transitions.....	315
Figure 7.5	PPS Negative Voltage Transitions.....	316
Figure 7.6	SPR PPS Programmable Voltage and Current Limit.....	318
Figure 7.7	SPR PPS Constant Power.....	319
Figure 7.8	AVS Positive Voltage Transitions	321
Figure 7.9	AVS Negative Voltage Transitions.....	321
Figure 7.10	Source VBUS and VCONN Response to Hard Reset.....	323
Figure 7.11	Application of vSrcNew and vSrcValid limits after tSrcReady.....	325
Figure 7.12	Expected AVS/PPS Ripple Relative to an LSB.....	326
Figure 7.13	Allowed DNL errors and tolerance of Voltage and Current in AVS/PPS mode	327
Figure 7.14	Source Peak Current Overload	329
Figure 7.15	Holdup Time Measurement	330
Figure 7.16	VBUS Power during Fast Role Swap	331
Figure 7.17	VBUS detection and timing during Fast Role Swap, initial VBUS (at new source) > vSafe5V(min).....	332
Figure 7.18	VBUS detection and timing during Fast Role Swap, initial VBUS (at new source) < vSafe5V(min).....	332
Figure 7.19	Data Reset UFP VCONN Power Cycle	333
Figure 7.20	Data Reset DFP VCONN Power Cycle	334
Figure 7.21	Placement of Sink Bulk Capacitance	335
Figure 7.22	Transition Diagram for Increasing the Voltage.....	342
Figure 7.23	Transition Diagram for Increasing the Voltage and Current.....	345
Figure 7.24	Transition Diagram for Increasing the Voltage and Decreasing the Current.....	348
Figure 7.25	Transition Diagram for Decreasing the Voltage and Increasing the Current.....	351
Figure 7.26	Transition Diagram for Decreasing the Voltage.....	354
Figure 7.27	Transition Diagram for Decreasing the Voltage and the Current.....	357
Figure 7.28	Transition Diagram for no change in Current or Voltage	360
Figure 7.29	Transition Diagram for Increasing the Current	362
Figure 7.30	Transition Diagram for Decreasing the Current.....	365
Figure 7.31	Transition Diagram for Increasing the Programmable Power Supply Voltage.....	368
Figure 7.32	Transition Diagram for Decreasing the Programmable Power Supply Voltage	371
Figure 7.33	Transition Diagram for increasing the Current in PPS mode.....	374
Figure 7.34	Transition Diagram for decreasing the Current in PPS mode.....	377
Figure 7.35	Transition Diagram for no change in Current or Voltage in PPS mode	380
Figure 7.36	Transition Diagram for Increasing the Adjustable Power Supply Voltage.....	382
Figure 7.37	Transition Diagram for Decreasing the Adjustable Voltage Supply Voltage.....	385
Figure 7.38	Transition Diagram for no change in Current or Voltage in AVS mode.....	387
Figure 7.39	Transition Diagram for a Sink Requested Power Role Swap	389
Figure 7.40	Transition Diagram for a Source Requested Power Role Swap	393
Figure 7.41	Transition Diagram for a Source Initiated Hard Reset.....	397
Figure 7.42	Transition Diagram for a Sink Initiated Hard Reset.....	399
Figure 7.43	Transition Diagram for Fast Role Swap.....	401

Figure 8.1	Example of daisy chained displays	422
Figure 8.2	Basic Message Exchange (Successful)	426
Figure 8.3	Basic Message flow indicating possible errors.....	427
Figure 8.4	Basic Message Flow with Bad followed by a Retry.....	428
Figure 8.5	Successful Fixed, Variable or Battery SPR Power Negotiation	448
Figure 8.6	Rejected Fixed, Variable or Battery SPR Power Negotiation.....	452
Figure 8.7	Wait response to Fixed, Variable or Battery SPR Power Negotiation	455
Figure 8.8	SPR PPS Keep Alive.....	458
Figure 8.9	SPR Sink Makes Request (Accept)	461
Figure 8.10	SPR Sink Makes Request (Reject).....	464
Figure 8.11	SPR Sink Makes Request (Wait).....	467
Figure 8.12	Entering EPR Mode (Success).....	470
Figure 8.13	Entering EPR Mode (Failure due to non-EPR cable).....	473
Figure 8.14	Entering EPR Mode (Failure of VCONN Swap).....	476
Figure 8.15	Successful Fixed EPR Power Negotiation	480
Figure 8.16	Rejected Fixed EPR Power Negotiation.....	484
Figure 8.17	Wait response to Fixed EPR Power Negotiation.....	487
Figure 8.18	EPR Keep Alive	490
Figure 8.19	Exiting EPR Mode (Sink Initiated)	493
Figure 8.20	Exiting EPR Mode (Source Initiated)	496
Figure 8.21	EPR Sink Makes Request (Accept)	499
Figure 8.22	EPR Sink Makes Request (Reject)	502
Figure 8.23	EPR Sink Makes Request (Wait)	505
Figure 8.24	Unsupported message.....	508
Figure 8.25	Soft Reset	511
Figure 8.26	DFP Initiated Data Reset where the DFP is the VCONN Source.....	514
Figure 8.27	DFP Receives Data Reset where the DFP is the VCONN Source	517
Figure 8.28	DFP Initiated Data Reset where the UFP is the VCONN Source.....	520
Figure 8.29	DFP Receives a Data Reset where the UFP is the VCONN Source.....	524
Figure 8.30	Source initiated Hard Reset.....	528
Figure 8.31	Sink Initiated Hard Reset.....	531
Figure 8.32	Source initiated reset - Sink long reset.....	534
Figure 8.33	Successful Power Role Swap Sequence Initiated by the Source	538
Figure 8.34	Rejected Power Role Swap Sequence Initiated by the Source.....	542
Figure 8.35	Power Role Swap Sequence with wait Initiated by the Source	545
Figure 8.36	Successful Power Role Swap Sequence Initiated by the Sink	549
Figure 8.37	Rejected Power Role Swap Sequence Initiated by the Sink.....	553
Figure 8.38	Power Role Swap Sequence with wait Initiated by the Sink	556
Figure 8.39	Successful Fast Role Swap Sequence	560
Figure 8.40	Data Role Swap, UFP operating as Sink initiates.....	564
Figure 8.41	Rejected Data Role Swap, UFP operating as Sink initiates.....	567
Figure 8.42	Data Role Swap with Wait, UFP operating as Sink initiates.....	570
Figure 8.43	Data Role Swap, UFP operating as Source initiates	573
Figure 8.44	Rejected Data Role Swap, UFP operating as Source initiates.....	576
Figure 8.45	Data Role Swap with Wait, UFP operating as Source initiates	579
Figure 8.46	Data Role Swap, DFP operating as Source initiates.....	582
Figure 8.47	Rejected Data Role Swap, DFP operating as Source initiates.....	585
Figure 8.48	Data Role Swap with Wait, DFP operating as Source initiates	588
Figure 8.49	Data Role Swap, DFP operating as Sink initiates.....	591
Figure 8.50	Rejected Data Role Swap, DFP operating as Sink initiates.....	594
Figure 8.51	Data Role Swap with Wait, DFP operating as Sink initiates.....	597
Figure 8.52	Successful VCONN Source Swap, initiated by VCONN Source	600
Figure 8.53	Rejected VCONN Source Swap, initiated by VCONN Source.....	603
Figure 8.54	VCONN Source Swap with Wait, initiated by VCONN Source.....	606
Figure 8.55	VCONN Source Swap, initiated by non-VCONN Source.....	609
Figure 8.56	Rejected VCONN Source Swap, initiated by non-VCONN Source.....	612

Figure 8.57	VCONN Source Swap with Wait.....	615
Figure 8.58	Source Alert to Sink.....	618
Figure 8.59	Sink Alert to Source.....	620
Figure 8.60	Sink Gets Source Status.....	622
Figure 8.61	Source Gets Sink Status.....	625
Figure 8.62	VCONN Source Gets Cable Plug Status.....	628
Figure 8.63	Sink Gets Source PPS Status.....	631
Figure 8.64	Sink Gets Source's Capabilities.....	634
Figure 8.65	Dual-Role Source Gets Dual-Role Sink's Capabilities as a Source	637
Figure 8.66	Source Gets Sink's Capabilities.....	640
Figure 8.67	Dual-Role Sink Gets Dual-Role Source's Capabilities as a Sink	643
Figure 8.68	Sink Gets Source's EPR Capabilities	646
Figure 8.69	Dual-Role Source Gets Dual-Role Sink's Capabilities as an EPR Source.....	649
Figure 8.70	Source Gets Sink's EPR Capabilities	652
Figure 8.71	Dual-Role Sink Gets Dual-Role Source's Capabilities as an EPR Sink	655
Figure 8.72	Sink Gets Source's Extended Capabilities	658
Figure 8.73	Dual-Role Source Gets Dual-Role Sink's Extended Capabilities	661
Figure 8.74	Source Gets Sink's Extended Capabilities	664
Figure 8.75	Dual-Role Sink Gets Dual-Role Source's Extended Capabilities	667
Figure 8.76	Sink Gets Source's Battery Capabilities	670
Figure 8.77	Source Gets Sink's Battery Capabilities	673
Figure 8.78	Sink Gets Source's Battery Status.....	676
Figure 8.79	Source Gets Sink's Battery Status.....	679
Figure 8.80	Source Gets Sink's Port Manufacturer Information	682
Figure 8.81	Sink Gets Source's Port Manufacturer Information	685
Figure 8.82	Source Gets Sink's Battery Manufacturer Information.....	688
Figure 8.83	Sink Gets Source's Battery Manufacturer Information.....	691
Figure 8.84	VCONN Source Gets Cable Plug's Manufacturer Information.....	694
Figure 8.85	Source Gets Sink's Country Codes.....	697
Figure 8.86	Sink Gets Source's Country Codes.....	700
Figure 8.87	VCONN Source Gets Cable Plug's Country Codes.....	703
Figure 8.88	Source Gets Sink's Country Information	706
Figure 8.89	Sink Gets Source's Country Information	709
Figure 8.90	VCONN Source Gets Cable Plug's Country Information	712
Figure 8.91	Source Gets Sink's Revision Information	715
Figure 8.92	Sink Gets Source's Revision Information	718
Figure 8.93	VCONN Source Gets Cable Plug's Revision Information	721
Figure 8.94	Sink Gets Source's Information.....	724
Figure 8.95	Dual-Role Source Gets Dual-Role Sink's Information as a Source.....	727
Figure 8.96	Source requests security exchange with Sink.....	730
Figure 8.97	Sink requests security exchange with Source.....	733
Figure 8.98	VCONN Source requests security exchange with Cable Plug.....	736
Figure 8.99	Source requests firmware update exchange with Sink	739
Figure 8.100	Sink requests firmware update exchange with Source	742
Figure 8.101	VCONN Source requests firmware update exchange with Cable Plug.....	745
Figure 8.102	Initiator to Responder Discover Identity (ACK)	748
Figure 8.103	Initiator to Responder Discover Identity (NAK).....	751
Figure 8.104	Initiator to Responder Discover Identity (BUSY).....	754
Figure 8.105	Initiator to Responder Discover SVIDs (ACK)	757
Figure 8.106	Initiator to Responder Discover SVIDs (NAK).....	760
Figure 8.107	Initiator to Responder Discover SVIDs (BUSY)	763
Figure 8.108	Initiator to Responder Discover Modes (ACK)	766
Figure 8.109	Initiator to Responder Discover Modes (NAK)	769
Figure 8.110	Initiator to Responder Discover Modes (BUSY).....	772
Figure 8.111	DFP to UFP Enter Mode	775
Figure 8.112	DFP to UFP Exit Mode.....	778
Figure 8.113	DFP to Cable Plug Enter Mode	781
Figure 8.114	DFP to Cable Plug Exit Mode	784

Figure 8.115	Initiator to Responder Attention.....	787
Figure 8.116	BIST Carrier Mode Test.....	789
Figure 8.117	BIST Test Data Test.....	792
Figure 8.118	BIST Share Capacity Mode Test.....	795
Figure 8.119	UFP Entering USB4 Mode (Accept).....	798
Figure 8.120	UFP Entering USB4 Mode (Reject).....	801
Figure 8.121	UFP Entering USB4 Mode (Wait).....	804
Figure 8.122	Cable Plug Entering USB4 Mode (Accept).....	807
Figure 8.123	Cable Plug Entering USB4 Mode (Reject).....	810
Figure 8.124	Cable Plug Entering USB4 Mode (Wait).....	813
Figure 8.125	Unstructured VDM Message Sequence.....	816
Figure 8.126	VDEM Message Sequence.....	819
Figure 8.127	Outline of States.....	822
Figure 8.128	References to states.....	823
Figure 8.129	Example of state reference with conditions.....	823
Figure 8.130	Example of state reference with the same entry and exit.....	823
Figure 8.131	SenderResponseTimer Policy Engine State Diagram.....	825
Figure 8.132	Source Port State Diagram.....	827
Figure 8.133	Sink Port State Diagram.....	835
Figure 8.134	SOP Source Port Soft Reset and Protocol Error State Diagram.....	841
Figure 8.135	Sink Port Soft Reset and Protocol Error Diagram.....	843
Figure 8.136	DFP Data_Reset Message State Diagram.....	845
Figure 8.137	UFP Data_Reset Message State Diagram.....	847
Figure 8.138	Source Port Not Supported Message State Diagram.....	850
Figure 8.139	Sink Port Not Supported Message State Diagram.....	852
Figure 8.140	Source Port Source Alert State Diagram.....	854
Figure 8.141	Sink Port Source Alert State Diagram.....	855
Figure 8.142	Sink Port Sink Alert State Diagram.....	856
Figure 8.143	Source Port Sink Alert State Diagram.....	857
Figure 8.144	Sink Port Get Source Capabilities Extended State Diagram.....	858
Figure 8.145	Source Give Source Capabilities Extended State Diagram.....	859
Figure 8.146	Source Port Get Sink Capabilities Extended State Diagram.....	860
Figure 8.147	Sink Give Sink Capabilities Extended State Diagram.....	861
Figure 8.148	Sink Port Get Source Information State Diagram.....	862
Figure 8.149	Source Give Source Information State Diagram.....	863
Figure 8.150	Get Status State Diagram.....	864
Figure 8.151	Give Status State Diagram.....	865
Figure 8.152	Sink Port Get Source PPS Status State Diagram.....	866
Figure 8.153	Source Give Source PPS Status State Diagram.....	867
Figure 8.154	Get Battery Capabilities State Diagram.....	868
Figure 8.155	Give Battery Capabilities State Diagram.....	869
Figure 8.156	Get Battery Status State Diagram.....	870
Figure 8.157	Give Battery Status State Diagram.....	871
Figure 8.158	Get Manufacturer Information State Diagram.....	872
Figure 8.159	Give Manufacturer Information State Diagram.....	873
Figure 8.160	Get Country Codes State Diagram.....	874
Figure 8.161	Give Country Codes State Diagram.....	875
Figure 8.162	Get Country Information State Diagram.....	876
Figure 8.163	Give Country Information State Diagram.....	877
Figure 8.164	Get Revision State Diagram.....	878
Figure 8.165	Give Revision State Diagram.....	879
Figure 8.166	DFP Enter_USB Message State Diagram.....	880
Figure 8.167	UFP Enter_USB Message State Diagram.....	881
Figure 8.168	Send security request State Diagram.....	882
Figure 8.169	Send security response State Diagram.....	883
Figure 8.170	Security response received State Diagram.....	884
Figure 8.171	Send firmware update request State Diagram.....	885
Figure 8.172	Send firmware update response State Diagram.....	886

Figure 8.173	Firmware update response received State Diagram.....	887
Figure 8.174	DFP to UFP Data Role Swap State Diagram.....	888
Figure 8.175	UFP to DFP Data Role Swap State Diagram.....	891
Figure 8.176	Dual-Role Port in Source to Sink Power Role Swap State Diagram.....	894
Figure 8.177	Dual-role Port in Sink to Source Power Role Swap State Diagram.....	898
Figure 8.178	Dual-Role Port in Source to Sink Fast Role Swap State Diagram.....	902
Figure 8.179	Dual-role Port in Sink to Source Fast Role Swap State Diagram.....	906
Figure 8.180	Dual-Role (Source) Get Source Capabilities diagram	909
Figure 8.181	Dual-Role (Source) Give Sink Capabilities diagram	910
Figure 8.182	Dual-Role (Sink) Get Sink Capabilities State Diagram.....	911
Figure 8.183	Dual-Role (Sink) Give Source Capabilities State Diagram	912
Figure 8.184	Dual-Role (Source) Get Source Capabilities Extended State Diagram	913
Figure 8.185	Dual-Role (Sink) Give Source Capabilities Extended diagram.....	914
Figure 8.186	Dual-Role (Sink) Get Sink Capabilities Extended State Diagram.....	915
Figure 8.187	Dual-Role (Source) Give Sink Capabilities Extended diagram.....	916
Figure 8.188	Dual-Role (Source) Get Source Information State Diagram	917
Figure 8.189	Dual-Role (Source) Give Source Information diagram	918
Figure 8.190	VCONN Swap State Diagram	919
Figure 8.191	Initiator to Port VDM Discover Identity State Diagram.....	922
Figure 8.192	Initiator VDM Discover SVIDs State Diagram	924
Figure 8.193	Initiator VDM Discover Modes State Diagram.....	926
Figure 8.194	Initiator VDM Attention State Diagram	928
Figure 8.195	Responder Structured VDM Discover Identity State Diagram	929
Figure 8.196	Responder Structured VDM Discover SVIDs State Diagram.....	930
Figure 8.197	Responder Structured VDM Discover Modes State Diagram	931
Figure 8.198	Receiving a Structured VDM Attention State Diagram	932
Figure 8.199	DFP VDM Mode Entry State Diagram	933
Figure 8.200	DFP VDM Mode Exit State Diagram.....	935
Figure 8.201	UFP Structured VDM Enter Mode State Diagram.....	937
Figure 8.202	UFP Structured VDM Exit Mode State Diagram.....	939
Figure 8.203	Cable Ready State Diagram	941
Figure 8.204	Cable Plug Soft Reset State Diagram	942
Figure 8.205	Cable Plug Hard Reset State Diagram.....	943
Figure 8.206	DFP/VCONN Source Soft Reset or Cable Reset of a Cable Plug or VPD State Diagram.....	944
Figure 8.207	UFP/VCONN Source Soft Reset of a Cable Plug or VPD State Diagram	946
Figure 8.208	Source Startup Structured VDM Discover Identity State Diagram	947
Figure 8.209	Cable Plug Structured VDM Enter Mode State Diagram.....	949
Figure 8.210	Cable Plug Structured VDM Exit Mode State Diagram	951
Figure 8.211	Source EPR Mode Entry State Diagram	953
Figure 8.212	Sink EPR Mode Entry State Diagram.....	956
Figure 8.213	Source EPR Mode Exit State Diagram.....	958
Figure 8.214	Sink EPR Mode Exit State Diagram	959
Figure 8.215	BIST Carrier Mode State Diagram	960
Figure 8.216	BIST Test Data Mode State Diagram	962
Figure 8.217	BIST Shared Capacity Test Mode State Diagram	964

9 States and Status Reporting 975

Figure 9.1	Example PD Topology	976
Figure 9.2	Mapping of PD Topology to USB	977
Figure 9.3	Use of SPM in the PD System	978
Figure 9.4	USB Attached to USB Powered State Transition.....	980
Figure 9.5	Any USB State to USB Attached State Transition (When operating as a Consumer)	981
Figure 9.6	Any USB State to USB Attached State Transition (When operating as a Provider)	981
Figure 9.7	Any USB State to USB Attached State Transition (After a USB Type-C Data Role Swap).....	982
Figure 9.8	Software stack on a PD aware OS	983
Figure 9.9	Enumeration of a PDUSB Device.....	984

10	Power Rules	995
Figure 10.1	SPR Source Power Rule Illustration for Fixed Supply PDOs.....	1000
Figure 10.2	SPR Source Power Rule Example For Fixed Supply PDOs.....	1001
Figure 10.3	Valid SPR AVS Operating Region for a Source advertising in the range of $27W < PDP \leq 45W$ 1003	
Figure 10.4	Valid SPR AVS Operating Region for a Source advertising in the range of $45W < PDP \leq 60W$ 1003	
Figure 10.5	Valid SPR AVS Operating Region for a Source advertising in the range of $60W < PDP \leq$ 100W.....	1004
Figure 10.6	Valid EPR AVS Operating Region.....	1011
Figure 10.7	EPR Source Power Rule Illustration for Fixed PDOs.....	1012
A	CRC calculation	1015
B	Message Sequence Examples (Deprecated)	1016
C	VDM Command Examples	1017
D	BMC Receiver Design Examples	1032
Figure D.1	Circuit Block of BMC Finite Difference Receiver.....	1032
Figure D.2	BMC AC and DC noise from VBUS at Power Sink.....	1033
Figure D.3	Sample BMC Signals (a) without USB 2.0 SE0 Noise (b) with USB 2.0 SE0 Noise.....	1033
Figure D.4	Scaled BMC Signal Derivative with 50ns Sampling Rate (a) without USB 2.0 Noise (b) with USB 2.0 Noise	1034
Figure D.5	BMC Signal and Finite Difference Output with Various Time Steps.....	1034
Figure D.6	Output of Finite Difference in dash line and Edge Detector in solid line	1035
Figure D.7	Noise Zone and Detect Zone of BMC Receiver	1035
Figure D.8	Circuit Block of BMC Subtraction Receiver	1036
Figure D.9	(a) Output of LPF1 and LPF2 (b) Subtraction of LPF1 and LPF2 Output.....	1036
Figure D.10	Output of the BMC LPF1 in blue dash curve and the Subtractor in red solid curve (a) at Power Source (b) at Power Sink.....	1037
E	FRS System Level Example	1038
Figure E.1	Example FRS Capable System.....	1038
Figure E.2	Slow VBUS Discharge	1039
Figure E.3	Fast VBUS Discharge.....	1040
Figure E.4	Slow VBUS discharge after FR_Swap message is sent.....	1044
Figure E.5	VBUS discharges quickly before FR_Swap message is sent after adapter disconnected.....	1046

List of Tables

Revision History	2
LIMITED COPYRIGHT LICENSE	5
INTELLECTUAL PROPERTY DISCLAIMER	5
Editors	6
Contributors	6
Table Of Contents	13
List of Figures	19
List of Tables	27
1 Introduction	34
Table 1.1 Section Overview	36
Table 1.2 Keywords	37
Table 1.3 Document References	39
Table 1.4 Terms and Abbreviations	40
2 Overview	53
Table 2.1 Fixed Supply Power Ranges	75
Table 2.2 PPS Voltage Power Ranges	75
Table 2.3 AVS Voltage Power Ranges	76
3 USB Type-A and USB Type-B Cable Assemblies and Connectors	77
4 Electrical Requirements	78
5 Physical Layer	79
Table 5.1 4b5b Symbol Encoding	80
Table 5.2 Ordered Sets	81
Table 5.3 Validation of Ordered Sets	82
Table 5.4 Data Size	83
Table 5.5 SOP Ordered Set	84
Table 5.6 SOP' Ordered Set	85
Table 5.7 SOP'' Ordered Set	85
Table 5.8 SOP'_Debug Ordered Set	86
Table 5.9 SOP''_Debug Ordered Set	86
Table 5.10 CRC-32 Mapping	88
Table 5.11 Hard Reset Ordered Set	89
Table 5.12 Cable Reset Ordered Set	90
Table 5.13 R_p values used for Collision Avoidance	91
Table 5.14 BMC Tx Mask Definition, X Values	97
Table 5.15 BMC Tx Mask Definition, Y Values	98
Table 5.16 BMC Rx Mask Definition	102
Table 5.17 BMC Common Normative Requirements	105
Table 5.18 BMC Transmitter Normative Requirements	106
Table 5.19 BMC Receiver Normative Requirements	110
6 Protocol Layer	114
Table 6.1 Message Header	116
Table 6.2 Revision Interoperability during an Explicit Contract	119
Table 6.3 Extended Message Header	120
Table 6.4 Use of Unchunked Message Supported bit	122
Table 6.5 Control Message Types	128

Table 6.6	Data Message Types	138
Table 6.7	Power Data Object.....	139
Table 6.8	Augmented Power Data Object.....	140
Table 6.9	Fixed Supply PDO – Source	141
Table 6.10	Fixed Power Source Peak Current Capability	143
Table 6.11	Variable Supply (non-Battery) PDO – Source	143
Table 6.12	Battery Supply PDO – Source	144
Table 6.13	SPR Programmable Power Supply APDO – Source.....	145
Table 6.14	SPR Adjustable Voltage Supply APDO – Source.....	146
Table 6.15	EPR Adjustable Voltage Supply APDO – Source	146
Table 6.16	EPR AVS Power Source Peak Current Capability	147
Table 6.17	Fixed Supply PDO – Sink.....	148
Table 6.18	Variable Supply (non-Battery) PDO – Sink.....	149
Table 6.19	Battery Supply PDO – Sink.....	150
Table 6.20	SPR Programmable Power Supply APDO – Sink.....	150
Table 6.21	SPR Adjustable Voltage Supply APDO – Sink.....	151
Table 6.22	EPR Adjustable Voltage Supply APDO – Sink	151
Table 6.23	Fixed and Variable Request Data Object	155
Table 6.24	Battery Request Data Object.....	156
Table 6.25	PPS Request Data Object	156
Table 6.26	AVS Request Data Object.....	156
Table 6.27	BIST Data Object	160
Table 6.28	Unstructured VDM Header.....	163
Table 6.29	Structured VDM Header	164
Table 6.30	Structured VDM Commands	165
Table 6.31	SVID Values	165
Table 6.32	Commands and Responses.....	167
Table 6.33	ID Header VDO.....	169
Table 6.34	Product Types (UFP).....	170
Table 6.35	Product Types (Cable Plug/VPD).....	171
Table 6.36	Product Types (DFP).....	171
Table 6.37	Cert Stat VDO	172
Table 6.38	Product VDO	172
Table 6.39	UFP VDO	172
Table 6.40	DFP VDO	175
Table 6.41	Passive Cable VDO.....	176
Table 6.42	Active Cable VDO1.....	179
Table 6.43	Active Cable VDO2.....	180
Table 6.44	VPD VDO	184
Table 6.45	Discover SVIDs Responder VDO.....	186
Table 6.46	Battery Status Data Object (BSDO).....	196
Table 6.47	Alert Data Object (ADO).....	198
Table 6.48	Country Code Data Object (CCDO)	201
Table 6.49	Enter_USB Data Object (EUDO)	203
Table 6.50	EPR Mode Data Object (EPRMDO)	206
Table 6.51	Source_Info Data Object (SIDO).....	212
Table 6.52	Revision Message Data Object (RMDO)	214
Table 6.53	Extended Message Types	215
Table 6.54	Source Capabilities Extended Data Block (SCEDB)	216
Table 6.55	SOP Status Data Block (SDB).....	221
Table 6.56	“SOP’/SOP” Status Data Block (SPDB)”	226
Table 6.57	Get Battery Cap Data Block (GBCDB).....	227
Table 6.58	Get Battery Status Data Block (GBSDB).....	227
Table 6.59	Battery Capability Data Block (BCDB)”	228
Table 6.60	Get Manufacturer Info Data Block (GMIDB)	230
Table 6.61	Manufacturer Info Data Block (MIDB)	231
Table 6.62	PPS Status Data Block (PPSSDB).....	235
Table 6.63	Country Codes Data Block (CCDB).....	237

Table 6.64	Country Info Data Block (CIDB)	238
Table 6.65	Sink Capabilities Extended Data Block (SKEDB)	239
Table 6.66	Extended Control Data Block (ECDB)	244
Table 6.67	Extended Control Message Types	244
Table 6.68	Time Values	262
Table 6.69	Timers	264
Table 6.70	Counter Parameters	267
Table 6.71	Counters	267
Table 6.72	Response to an incoming Message (except VDM)	269
Table 6.73	Response to an incoming VDM	269
Table 6.74	Message Discarding	274
Table 6.75	Protocol Layer States	298
Table 6.76	Message Applicability Abbreviations	300
Table 6.77	Applicability of Control Messages	301
Table 6.78	Applicability of Data Messages	303
Table 6.79	Applicability of Extended Messages	305
Table 6.80	Applicability of Extended Control Messages	308
Table 6.81	Applicability of Structured VDM Commands	309
Table 6.82	Applicability of Reset Signaling	310
Table 6.83	Applicability of Fast Role Swap Request	310
Table 6.84	Value Parameters	311

7 Power Supply 312

Table 7.1	Sequence Description for Increasing the Voltage	343
Table 7.2	Sequence Diagram for Increasing the Voltage and Current	346
Table 7.3	Sequence Description for Increasing the Voltage and Decreasing the Current	349
Table 7.4	Sequence Description for Decreasing the Voltage and Increasing the Current	352
Table 7.5	Sequence Description for Decreasing the Voltage	355
Table 7.6	Sequence Description for Decreasing the Voltage and the Current	358
Table 7.7	Sequence Description for no change in Current or Voltage	360
Table 7.8	Sequence Description for Increasing the Current	363
Table 7.9	Sequence Description for Decreasing the Current	366
Table 7.10	Sequence Description for Increasing the Programmable Power Supply Voltage	369
Table 7.11	Sequence Description for Decreasing the Programmable Power Supply Voltage	372
Table 7.12	Sequence Description for increasing the Current in PPS mode	375
Table 7.13	Sequence Description for decreasing the Current in PPS mode	378
Table 7.14	Sequence Description for no change in Current or Voltage in PPS mode	380
Table 7.15	Sequence Description for Increasing the Adjustable Voltage Supply Voltage	383
Table 7.16	Sequence Description for Decreasing the Adjustable Voltage Supply Voltage	386
Table 7.17	Sequence Description for no change in Current or Voltage in AVS mode	387
Table 7.18	Sequence Description for a Sink Requested Power Role Swap	390
Table 7.19	Sequence Description for a Source Requested Power Role Swap	394
Table 7.20	Sequence Description for a Source Initiated Hard Reset	397
Table 7.21	Sequence Description for a Sink Initiated Hard Reset	399
Table 7.22	Sequence Description for Fast Role Swap	402
Table 7.23	Source Electrical Parameters	404
Table 7.24	Sink Electrical Parameters	413
Table 7.25	Common Source/Sink Electrical Parameters	415

8 Device Policy 416

Table 8.1	Basic Message Flow	426
Table 8.2	Potential issues in Basic Message Flow	427
Table 8.3	Basic Message Flow with CRC failure	429
Table 8.4	Atomic Message Sequences	430
Table 8.5	AMS: Power Negotiation (SPR)	431
Table 8.6	AMS: Power Negotiation (EPR)	432
Table 8.7	AMS: Unsupported Message	434

Table 8.8	AMS: Soft Reset.....	434
Table 8.9	AMS: Data Reset	434
Table 8.10	AMS: Power Role Swap	435
Table 8.11	AMS: Fast Role Swap	435
Table 8.12	AMS: Data Role Swap	436
Table 8.13	AMS: VCONN Swap	437
Table 8.14	AMS: Alert	437
Table 8.15	AMS: Status.....	438
Table 8.16	AMS: Source/Sink Capabilities (SPR)	438
Table 8.17	AMS: Source/Sink Capabilities (EPR).....	439
Table 8.18	AMS: Extended Capabilities	440
Table 8.19	AMS: Battery Capabilities.....	440
Table 8.20	AMS: Manufacturer Information.....	441
Table 8.21	AMS: Country Codes	441
Table 8.22	AMS: Country Information	441
Table 8.23	AMS: Revision Information	442
Table 8.24	AMS: Source Information	442
Table 8.25	AMS: Security	442
Table 8.26	AMS: Firmware Update.....	443
Table 8.27	AMS: Structured VDM.....	444
Table 8.28	AMS: Built-In Self-Test (BIST).....	445
Table 8.29	AMS: Enter USB	445
Table 8.30	AMS: Unstructured VDM	445
Table 8.31	AMS: Hard Reset	446
Table 8.32	Steps for a successful Power Negotiation	449
Table 8.33	Steps for a rejected Power Negotiation	453
Table 8.34	Steps for a Wait response to a Power Negotiation.....	456
Table 8.35	Steps for SPR PPS Keep Alive	459
Table 8.36	Steps for SPR Sink Makes Request (Accept)	462
Table 8.37	Steps for SPR Sink Makes Request (Reject)	465
Table 8.38	Steps for SPR Sink Makes Request (Wait)	468
Table 8.39	Steps for Entering EPR Mode (Success).....	471
Table 8.40	Steps for Entering EPR Mode (Failure due to non-EPR cable)	474
Table 8.41	Steps for Entering EPR Mode (Failure of VCONN Swap)	477
Table 8.42	Steps for a successful EPR Power Negotiation	481
Table 8.43	Steps for a Rejected EPR Power Negotiation.....	485
Table 8.44	Steps for a Wait response to an EPR Power Negotiation	488
Table 8.45	Steps for EPR Keep Alive.....	491
Table 8.46	Steps for Exiting EPR Mode (Sink Initiated)	494
Table 8.47	Steps for Exiting EPR Mode (Source Initiated).....	497
Table 8.48	Steps for EPR Sink Makes Request (Accept).....	500
Table 8.49	Steps for EPR Sink Makes Request (Reject).....	503
Table 8.50	Steps for SPR Sink Makes Request (Wait)	506
Table 8.51	Steps for an Unsupported Message	509
Table 8.52	Steps for a Soft Reset.....	512
Table 8.53	Steps for a DFP Initiated Data Reset where the DFP is the VCONN Source.....	515
Table 8.54	Steps for a DFP Receiving a Data Reset where the DFP is the VCONN Source	518
Table 8.55	Steps for a DFP Initiated Data Reset where the UFP is the VCONN Source.....	521
Table 8.56	Steps for a DFP Receiving a Data Reset where the UFP is the VCONN Source.....	525
Table 8.57	Steps for Source initiated Hard Reset.....	529
Table 8.58	Steps for Sink initiated Hard Reset.....	532
Table 8.59	Steps for Source initiated Hard Reset - Sink long reset	535
Table 8.60	Steps for a Successful Source Initiated Power Role Swap Sequence	539
Table 8.61	Steps for a Rejected Source Initiated Power Role Swap Sequence.....	543
Table 8.62	Steps for a Source Initiated Power Role Swap with Wait Sequence.....	546
Table 8.63	Steps for a Successful Sink Initiated Power Role Swap Sequence.....	550
Table 8.64	Steps for a Rejected Sink Initiated Power Role Swap Sequence	554
Table 8.65	Steps for a Sink Initiated Power Role Swap with Wait Sequence.....	557

Table 8.66	Steps for a Successful Fast Role Swap Sequence	561
Table 8.67	Steps for Data Role Swap, UFP operating as Sink initiates.....	565
Table 8.68	Steps for Rejected Data Role Swap, UFP operating as Sink initiates.....	568
Table 8.69	Steps for Data Role Swap with Wait, UFP operating as Sink initiates	571
Table 8.70	Steps for Data Role Swap, UFP operating as Source initiates.....	574
Table 8.71	Steps for Rejected Data Role Swap, UFP operating as Source initiates.....	577
Table 8.72	Steps for Data Role Swap with Wait, UFP operating as Source initiates	580
Table 8.73	Steps for Data Role Swap, DFP operating as Source initiates.....	583
Table 8.74	Steps for Rejected Data Role Swap, DFP operating as Source initiates	586
Table 8.75	Steps for Data Role Swap with Wait, DFP operating as Source initiates	589
Table 8.76	Steps for Data Role Swap, DFP operating as Sink initiates.....	592
Table 8.77	Steps for Rejected Data Role Swap, DFP operating as Sink initiates.....	595
Table 8.78	Steps for Data Role Swap with Wait, DFP operating as Sink initiates	598
Table 8.79	Steps for Source to Sink VCONN Source Swap	601
Table 8.80	Steps for Rejected VCONN Source Swap	604
Table 8.81	Steps for VCONN Source Swap with Wait.....	607
Table 8.82	Steps for VCONN Source Swap, Initiated by non-VCONN Source	610
Table 8.83	Steps for Rejected VCONN Source Swap, Initiated by non-VCONN Source.....	613
Table 8.84	Steps for VCONN Source Swap with Wait.....	616
Table 8.85	Steps for Source Alert to Sink.....	619
Table 8.86	Steps for Sink Alert to Source.....	621
Table 8.87	Steps for a Sink getting Source Status Sequence.....	623
Table 8.88	Steps for a Source getting Sink Status Sequence.....	626
Table 8.89	Steps for a VCONN Source getting Cable Plug Status Sequence.....	629
Table 8.90	Steps for a Sink getting Source PPS status Sequence.....	632
Table 8.91	Steps for a Sink getting Source Capabilities Sequence.....	635
Table 8.92	Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as a Source Sequence...	638
Table 8.93	Steps for a Source getting Sink Capabilities Sequence.....	641
Table 8.94	Steps for a Dual-Role Sink getting Dual-Role Source capabilities as a Sink Sequence	644
Table 8.95	Steps for a Sink getting EPR Source Capabilities Sequence	647
Table 8.96	Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as an EPR Source Sequence	650
Table 8.97	Steps for a Source getting Sink EPR Capabilities Sequence	653
Table 8.98	Steps for a Dual-Role Sink getting Dual-Role Source Capabilities as an EPR Sink Sequence	656
Table 8.99	Steps for a Sink getting Source Extended Capabilities Sequence	659
Table 8.100	Steps for a Dual-Role Source getting Dual-Role Sink Extended Capabilities Sequence.....	662
Table 8.101	Steps for a Source getting Sink Extended Capabilities Sequence	665
Table 8.102	Steps for a Dual-Role Sink getting Dual-Role Source Extended Capabilities Sequence.....	668
Table 8.103	Steps for a Sink getting Source Battery capabilities Sequence	671
Table 8.104	Steps for a Source getting Sink Battery capabilities Sequence	674
Table 8.105	Steps for a Sink getting Source Battery status Sequence	677
Table 8.106	Steps for a Source getting Sink Battery status Sequence	680
Table 8.107	Steps for a Source getting Sink's Port Manufacturer Information Sequence	683
Table 8.108	Steps for a Source getting Sink's Port Manufacturer Information Sequence	686
Table 8.109	Steps for a Source getting Sink's Battery Manufacturer Information Sequence.....	689
Table 8.110	Steps for a Source getting Sink's Battery Manufacturer Information Sequence.....	692
Table 8.111	Steps for a VCONN Source getting Sink's Port Manufacturer Information Sequence	695
Table 8.112	Steps for a Source getting Country Codes Sequence	698
Table 8.113	Steps for a Source getting Sink's Country Codes Sequence.....	701
Table 8.114	Steps for a VCONN Source getting Sink's Country Codes Sequence.....	704
Table 8.115	Steps for a Source getting Country Information Sequence	707
Table 8.116	Steps for a Source getting Sink's Country Information Sequence	710
Table 8.117	Steps for a VCONN Source getting Sink's Country Information Sequence	713
Table 8.118	Steps for a Source getting Revision Information Sequence	716
Table 8.119	Steps for a Source getting Sink's Revision Information Sequence.....	719
Table 8.120	Steps for a VCONN Source getting Sink's Revision Information Sequence.....	722
Table 8.121	Steps for a Sink getting Source Information Sequence	725

Table 8.122	Steps for a Dual-Role Source getting Dual-Role Sink's Information as a Source Sequence	728
Table 8.123	Steps for a Source requesting a security exchange with a Sink Sequence	731
Table 8.124	Steps for a Sink requesting a security exchange with a Source Sequence	734
Table 8.125	Steps for a VCONN Source requesting a security exchange with a Cable Plug Sequence	737
Table 8.126	Steps for a Source requesting a firmware update exchange with a Sink Sequence	740
Table 8.127	Steps for a Sink requesting a firmware update exchange with a Source Sequence	743
Table 8.128	Steps for a VCONN Source requesting a firmware update exchange with a Cable Plug Sequence	746
Table 8.129	Steps for Initiator to UFP Discover Identity (ACK)	749
Table 8.130	Steps for Initiator to UFP Discover Identity (NAK)	752
Table 8.131	Steps for Initiator to UFP Discover Identity (BUSY)	755
Table 8.132	Steps for DFP to UFP Discover SVIDs (ACK)	758
Table 8.133	Steps for DFP to UFP Discover SVIDs (NAK)	761
Table 8.134	Steps for DFP to UFP Discover SVIDs (BUSY)	764
Table 8.135	Steps for DFP to UFP Discover Modes (ACK)	767
Table 8.136	Steps for DFP to UFP Discover Modes (NAK)	770
Table 8.137	Steps for DFP to UFP Discover Modes (BUSY)	773
Table 8.138	Steps for DFP to UFP Enter Mode	776
Table 8.139	Steps for DFP to UFP Exit Mode	779
Table 8.140	Steps for DFP to Cable Plug Enter Mode	782
Table 8.141	Steps for DFP to Cable Plug Exit Mode	785
Table 8.142	Steps for Initiator to Responder Attention	788
Table 8.143	Steps for BIST Carrier Mode Test	790
Table 8.144	Steps for BIST Test Data Test	793
Table 8.145	Steps for BIST Shared Capacity Test Mode Test	796
Table 8.146	Steps for UFP USB4 Mode Entry (Accept)	799
Table 8.147	Steps for UFP USB4 Mode Entry (Reject)	802
Table 8.148	Steps for UFP USB4 Mode Entry (Wait)	805
Table 8.149	Steps for Cable Plug USB4 Mode Entry (Accept)	808
Table 8.150	Steps for Cable Plug USB4 Mode Entry (Reject)	811
Table 8.151	Steps for Cable Plug USB4 Mode Entry (Wait)	814
Table 8.152	Steps for Unstructured VDM Message Sequence	817
Table 8.153	Steps for VDEM Message Sequence	820
Table 8.154	Policy Engine States	966

9 States and Status Reporting 975

Table 9.1	USB Power Delivery Type Codes	985
Table 9.2	USB Power Delivery Capability Descriptor	986
Table 9.3	Battery Info Capability Descriptor	987
Table 9.4	PD Consumer Port Descriptor	988
Table 9.5	PD Provider Port Descriptor	989
Table 9.6	PD Requests	990
Table 9.7	PD Request Codes	990
Table 9.8	PD Feature Selectors	990
Table 9.9	Get Battery Status Request	991
Table 9.10	Battery Status Structure	991
Table 9.11	Set PD Feature	992
Table 9.12	Battery Wake Mask	993
Table 9.13	Charging Policy Encoding	994

10 Power Rules 995

Table 10.1	Considerations for Sources	996
Table 10.2	SPR Normative Voltages and Minimum Currents	997
Table 10.3	SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP	998
Table 10.4	SPR Source Port Present PDP less than Port Maximum PDP Examples	999
Table 10.5	Fixed Supply PDO - Source 5V	1001
Table 10.6	Fixed Supply PDO - Source 9V	1002

Table 10.7	Fixed Supply PDO - Source 15V	1002
Table 10.8	Fixed Supply PDO - Source 20V	1002
Table 10.9	SPR Adjustable Voltage Supply (AVS) Voltage Ranges.....	1004
Table 10.10	SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP...	1005
Table 10.11	SPR Programmable Power Supply Voltage Ranges	1006
Table 10.12	EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable	1009
Table 10.14	EPR Source Examples when Port Present PDP is less than Port Maximum PDP.....	1010
Table 10.13	EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable.....	1010
Table 10.15	EPR Adjustable Voltage Supply (AVS) Voltage Ranges	1012
A	CRC calculation	1015
B	Message Sequence Examples (Deprecated)	1016
C	VDM Command Examples	1017
Table C.1	Discover Identity Command request from Initiator Example	1017
Table C.2	Discover Identity Command response from Active Cable Responder Example	1018
Table C.3	Discover Identity Command response from Hub Responder Example	1020
Table C.4	Discover SVIDs Command request from Initiator Example.....	1021
Table C.5	Discover SVIDs Command response from Responder Example	1022
Table C.6	Discover Modes Command request from Initiator Example	1023
Table C.7	Discover Modes Command response from Responder Example.....	1024
Table C.8	Enter Mode Command request from Initiator Example	1025
Table C.9	Enter Mode Command response from Responder Example.....	1026
Table C.10	Enter Mode Command request with additional VDO Example	1027
Table C.11	Exit Mode Command request from Initiator Example	1028
Table C.12	Exit Mode Command response from Responder Example	1029
Table C.13	Attention Command request from Initiator Example.....	1030
Table C.14	Attention Command request from Initiator with additional VDO Example	1031
D	BMC Receiver Design Examples	1032
E	FRS System Level Example	1038
Table E.1	Sequence for setup of a Fast Role Swap (Hub connected to Power Adapter first)	1041
Table E.2	Sequence for setup of a Fast Role Swap (Hub connected to laptop before Power Adapter)	1042
Table E.3	Sequence for slow VBUS discharge (it discharges after FR_Swap message is sent)	1045
Table E.4	VBUS discharges quickly after adapter disconnected.....	1046

1 Introduction

USB has evolved from a data interface capable of supplying limited power to a primary provider of power with a data interface. Today many devices charge or get their power from USB Ports contained in laptops, cars, aircraft or even wall sockets. USB has become a ubiquitous power socket for many small devices such as cell phones and other hand-held devices. Users need USB to fulfill their requirements not only in terms of data but also to provide power to, or charge, their devices simply, often without the need to load a driver, in order to carry out “traditional” USB functions.

There are, however, still many devices which either require an additional power connection to the wall, or exceed the USB default current in order to operate. Increasingly, international regulations require better energy management due to ecological and practical concerns relating to the availability of power. Regulations limit the amount of power available from the wall which has led to a pressing need to optimize power usage. The USB Power Delivery Specification has the potential to minimize waste as it becomes a standard for charging devices that are not satisfied by [\[USBBC 1.2\]](#) or [\[USB Type-C 2.4\]](#).

Wider usage of wireless solutions is an attempt to remove data cabling but the need for “tethered” charging remains. In addition, industrial design requirements drive wired connectivity to do much more over the same connector.

USB Power Delivery is designed to enable the maximum functionality of USB by providing more flexible power delivery along with data over a single cable. Its aim is to operate with and build on the existing USB ecosystem; increasing power levels from existing USB standards, for example [\[USBBC 1.2\]](#), enabling new higher power use cases such as USB powered Hard Disk Drives (HDDs), laptops and monitors.

With USB Power Delivery the power direction is no longer fixed. This enables the product with the power (USB Host or Peripheral) to provide the power. For example, a display with a supply from the wall can power, or charge, a laptop. Alternatively, USB Chargers are able to supply power to laptops and other Battery powered devices through their, traditional power providing, USB Ports.

USB Power Delivery enables Hubs (including Hubs embedded in other devices such as docks or monitors) to become the means to optimize power management across multiple peripherals by allowing each device to take only the power it requires, and to get more power when required for a given application. **Optionally** the Hubs can communicate with the PC to enable even more intelligent and flexible management of power either automatically or with some level of user intervention.

USB Power Delivery allows low power cases such as headsets to Negotiate for only the power they require. This provides a simple solution that enables USB devices to operate at their optimal power levels.

The Power Delivery Specification, in addition to providing mechanisms to Negotiate power also can be used as a side-band channel for standard and vendor defined messaging. The specification enables discovery of cable Capabilities such as supported speeds and current levels. Power Delivery enables alternative modes of operation by providing the mechanisms to discover, enter and exit Modes such as EPR Mode, USB4[®] Mode or Alternate Modes.

1.1 Overview

This specification defines how USB Devices can Negotiate for more current and/or higher or lower voltages over the USB cable (using the USB Type-C[®] CC wire as the communications channel) than are defined in the [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB4\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#) specifications. It allows Devices with greater power requirements than can be met with today's specification to get the power they require to operate from *V_{BUS}* and Negotiate with external power sources (e.g., Chargers).

In addition, it allows a Source and Sink to swap Power Roles such that a USB Device could supply power to the USB Host. For example, a display could supply power to a laptop to operate or charge its Battery. This specification also adds a mechanism to swap the Data Roles such that the upstream facing Port becomes the downstream facing Port and vice versa. It also enables a swap of the end supplying *V_{CONN}* to a powered cable.

The USB Power Delivery Specification is guided by the following principles:

- Works seamlessly with legacy USB Devices.

- Compatible with existing spec-compliant USB cables.
- Minimizes potential damage from non-compliant cables (e.g., ‘Y’ cables etc.).
- Optimized for low-cost implementations.

This specification defines mechanisms to discover, enter and exit Alternate Modes defined either by a standard or by a particular vendor. These Alternate Modes can be supported either by the Port Partner or by a cable connecting the two Port Partners.

The specification defines mechanisms to discover the Capabilities of cables which can communicate using Power Delivery.

To facilitate optimum charging, the specification defines two mechanisms a USB Charger can Advertise for the device to use:

- 1) A list of Fixed Supply voltages each with a maximum current. The device selects a voltage and current from the list. This is the traditional model used by devices that use internal electronics to manage the charging of their Battery including modifying the voltage and current actually supplied to the Battery. The side-effect of this model is that the charging circuitry generates heat that can be problematic for small form factor devices.
- 2) A list of programmable voltage ranges, in SPR PPS Mode, each with a maximum current. The device requests a voltage (in 20mV increments) that is within the Advertised range and a maximum current. The USB PPS Charger delivers the requested voltage until the maximum current is reached at which time the USB PPS Charger reduces its output voltage so as not to supply more than the requested maximum current. During the high current portion of the charge cycle, the USB PPS Charger can be directly connected (through an appropriate safety device) to the Battery. This model is used by devices that want to minimize the thermal impact of their internal charging circuitry.
- 3) A list of adjustable voltage ranges, in SPR AVS Mode or EPR AVS Mode, each with a maximum current. The device requests a voltage (in 100mV increments) that is within the Advertised range and a maximum current. The USB AVS Charger delivers the requested voltage.

1.2 Purpose

The USB Power Delivery specification defines a power delivery system covering all elements of a USB system including USB Hosts, USB Devices, Hubs, Chargers and cable assemblies. This specification describes the architecture, protocols, power supply behavior, connectors and cabling necessary for managing power delivery over USB at up to 100W in SPR Mode and 240W in EPR Mode. This specification is intended to be fully compatible with and extend the existing USB infrastructure. It is intended that this specification will allow system OEMs, power supply and Peripheral developers adequate flexibility for product versatility and market differentiation without losing backwards compatibility.

USB Power Delivery is designed to operate independently of the existing USB bus defined mechanisms used to Negotiate power which are:

- [\[USB 2.0\]](#), [\[USB 3.2\]](#) in band requests for high power interfaces.
- [\[USBBC 1.2\]](#) mechanisms for supplying higher power (not mandated by this specification).
- [\[USB Type-C 2.4\]](#) mechanisms for supplying higher power.

Initial operating conditions remain the USB Default Operation as defined in [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#).

- The DFP sources **vSafe5V** over *VBUS*.
- The UFP consumes power from *VBUS*.

1.2.1 Scope

This specification is intended as an extension to the existing [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB Type-C 2.4\]](#) and [\[USBBC 1.2\]](#) specifications. It addresses only the elements required to implement USB Power Delivery. It is targeted at power supply vendors, manufacturers of [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB Type-C 2.4\]](#) and [\[USBBC 1.2\]](#) platforms, devices and cable assemblies.

Normative information is provided to allow interoperability of components designed to this specification. **Informative** information, when provided, illustrates possible design implementation.

1.3 Section Overview

This specification contains the following sections:

Table 1.1 Section Overview

Section	Description
Section 1, "Introduction"	Introduction, conventions used in the document, list of terms and abbreviations, references, and details of parameter usage.
Section 2, "Overview"	Overview of the document including a description of the operation of <i>PD</i> and the architecture.
Section 3, "USB Type-A and USB Type-B Cable Assemblies and Connectors"	Mechanical and electrical characteristics of the cables and connectors used by <i>PD</i> . Section Deprecated . See [USBPD 2.0] for legacy <i>PD</i> connector specification.
Section 4, "Electrical Requirements"	Electrical requirements for <i>Dead Battery</i> operation and cable detection.
Section 5, "Physical Layer"	Details of the <i>PD PHY Layer</i> requirements
Section 6, "Protocol Layer"	<i>Protocol Layer</i> requirements including the <i>Messages</i> , timers, counters, and state operation.
Section 7, "Power Supply"	Power supply requirements for both <i>Providers</i> and <i>Consumers</i> .
Section 8, "Device Policy"	<i>Device Policy Manager</i> requirements. <i>Policy Engine Atomic Message Sequence (AMS)</i> diagrams and state diagrams
Section 9, "States and Status Reporting"	<i>PDUSB Device</i> requirements including mapping of <i>VBUS</i> to USB states. <i>System Policy Manager</i> requirements including descriptors, events, and requests.
Section 10, "Power Rules"	<i>PDP Rating</i> definitions for <i>PD</i> .
Section A, "CRC calculation"	Example <i>CRC</i> calculations.
Section B, "Message Sequence Examples (Deprecated)"	Scenarios illustrating <i>Device Policy Manager</i> operation. Deprecated
Section C, "VDM Command Examples"	Examples of <i>Structured VDM</i> usage. Section Deprecated .
Section D, "BMC Receiver Design Examples"	<i>BMC Receiver Design</i> Examples.
Section E, "FRS System Level Example"	<i>FRS System Level</i> Example.

1.4 Conventions

1.4.1 Precedence

If there is a conflict between text, figures, and tables, the precedence **Shall** be tables, figures, and then text.

In there is a conflict between a generic statement and a more specific statement, the more specific statement **Shall** apply.

1.4.2 Keywords

The following keywords differentiate between the levels of requirements and options.

Table 1.2 Keywords

Keyword	Definition
Conditional Normative	Conditional Normative is a keyword used to indicate a feature that is mandatory when another related feature has been implemented. Designers are mandated to implement all such requirements, when the dependent features have been implemented, to ensure interoperability with other compliant devices.
Deprecated	Deprecated is a keyword used to indicate a feature, supported in previous releases of the specification, which is no longer supported.
Discard	See Discarded .
Discarded	Discard , Discards and Discarded are equivalent keywords indicating that a <i>Packet</i> when received Shall be thrown away by the <i>PHY Layer</i> and not passed to the <i>Protocol Layer</i> for processing. No GoodCRC Message Shall be sent in response to the <i>Packet</i> .
Discards	See Discarded .
Ignore	See Ignored .
Ignored	Ignore , Ignores and Ignored are equivalent keywords indicating <i>Messages</i> or <i>Message</i> fields which, when received, Shall result in no special action by the receiver. An Ignored Message Shall only result in returning a GoodCRC Message to acknowledge <i>Message</i> receipt. A <i>Message</i> with an Ignored field Shall be processed normally except for any actions relating to the Ignored field.
Ignores	See Ignored .
Informative	Informative is a keyword indicating text with no specific requirements, provided only to improve understanding.
Invalid	Invalid is a keyword when used in relation to a <i>Packet</i> indicates that the <i>Packet</i> 's usage or fields fall outside of the defined specification usage. When Invalid is used in relation to an <i>Explicit Contract</i> it indicates that a previously established <i>Explicit Contract</i> which can no longer be maintained by the <i>Source</i> . When Invalid is used in relation to individual <i>K-codes</i> or <i>K-code</i> sequences indicates that the received <i>Signaling</i> falls outside of the defined specification.
May	May is a keyword that indicates a choice with no implied preference.
May Not	May Not is a keyword that is the inverse of May . Indicates a choice to not implement a given feature with no implied preference.
N/A	N/A is a keyword that indicates that a field or value is not applicable and has no defined value and Shall Not be checked or used by the recipient.
Normative	See Shall .
Optional	Optional , Optionally and Optional Normative are equivalent keywords that describe features not mandated by this specification. However, if an Optional feature is implemented, the feature Shall be implemented as defined by this specification.
Optional Normative	See Optional .
Optionally	See Optional .

Table 1.2 Keywords (Continued)

Keyword	Definition
Reserved	Reserved is a keyword indicating bits, bytes, words, fields, and code values that are set-aside for future standardization. Their use and interpretation May be specified by future extensions to this specification and Shall Not be utilized or adapted by vendor implementation. A Reserved bit, byte, word, or field Shall be set to zero by the sender and Shall be Ignored by the receiver. Reserved field values Shall Not be sent by the sender and Shall be Ignored by the receiver.
Shall	Shall and Normative are equivalent keywords indicating a mandatory requirement. Designers are mandated to implement all such requirements to ensure interoperability with other compliant devices.
Shall Not	Shall Not is a keyword that is the inverse of Shall indicating non-compliant operation.
Should	Should is a keyword indicating flexibility of choice with a preferred alternative; equivalent to the phrase “it is recommended that...”.
Should Not	Should Not is a keyword is the inverse of Should ; equivalent to the phrase “it is recommended that implementations do not...”.
Static	Static is a keyword indicating that a field that never changes.
Valid	Valid is a keyword that is the inverse of Invalid indicating either a Packet or Signaling that fall within the defined specification or an <i>Explicit Contract</i> that can be maintained by the <i>Source</i> .

1.4.3 Numbering

Numbers that are immediately followed by a lowercase “b” (e.g., 01b) are binary values. Numbers that are immediately followed by an uppercase “B” are byte values. Numbers that are immediately followed by a lowercase “h” (e.g., 3Ah) or are preceded by “0x” (e.g., 0xFF00) are hexadecimal values. Numbers not immediately followed by either a “b”, “B”, or “h” are decimal values.

1.5 Related Documents

Document references listed in [Table 1.3, "Document References"](#) are inclusive of all approved and published ECNs and Errata.

Table 1.3 Document References

Bookmark Reference	Title
[DPTC2.1]	DisplayPort™ Alt Mode on USB Type-C Standard www.vesa.org .
[IEC 60950-1]	IEC 60950-1:2005 Information technology equipment – Safety – Part 1: General requirements: Amendment 1:2009, Amendment 2:2013. www.iec.ch .
[IEC 60958-1]	IEC 60958-1:2021 Digital Audio Interface Part:1 General. www.iec.ch .
[IEC 62368-1]	IEC 62368-1:2018 Audio/Video, information, and communication technology equipment – Part 1: Safety requirements. www.iec.ch .
[IEC 62368-3]	IEC 62368-3:2017 Audio/video, information, and communication technology equipment - Part 3: Safety aspects for DC power transfer through communication cables and ports www.iec.ch .
[IEC 63002]	IEC 63002:2021 Interoperability specifications and communication method for external power supplies used with computing and consumer electronics devices www.iec.ch .
[ISO 3166]	ISO 3166 international Standard for country codes and codes for their subdivisions. http://www.iso.org/iso/home/standards/country_codes.htm .
[TBT3]	see [USB4] Chapter 13 for Thunderbolt™ 3 device operation.
[UCSI]	USB Type-C Connector System Software Interface (UCSI) Specification https://www.usb.org/documents .
[USB 2.0]	Universal Serial Bus 2.0 Specification, https://www.usb.org/documents .
[USB 3.2]	Universal Serial Bus 3.2 Specification https://www.usb.org/documents .
[USB Type-C 2.4]	Universal Serial Bus Type-C Cable and Connector Specification, https://www.usb.org/documents .
[USB4]	Universal Serial Bus 4 Specification (USB4®), https://www.usb.org/documents .
[USBBC 1.2]	Universal Serial Bus Battery Charging Specification plus Errata (referred to in this document as the Battery Charging specification). https://www.usb.org/documents .
[USBPD 2.0]	Universal Serial Bus Power Delivery Specification, https://www.usb.org/documents .
[USBPDCompliance]	USB Power Delivery Compliance Test Specification, https://www.usb.org/documents .
[USBPDFirmwareUpdate 1.0]	Universal Serial Bus Power Delivery Firmware Update Specification, https://www.usb.org/documents .
[USBTypeCAuthentication 1.0]	Universal Serial Bus Type-C Authentication Specification, https://www.usb.org/documents .
[USBTypeCBridge 1.1]	Universal Serial Bus Type-C Bridge Specification, https://www.usb.org/documents .

1.6 Terms and Abbreviations

This section defines terms used throughout this document. For additional terms that pertain to the Universal Serial Bus, see Chapter 2, “Terms and Abbreviations,” in [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB Type-C 2.4\]](#) and [\[USBBC 1.2\]](#).

Table 1.4 Terms and Abbreviations

Term	Description
<i>(A)PDO</i>	Refers to both the <i>PDO</i> and <i>APDO</i> collectively.
<i>AC Supply</i> <i>AC Supplied</i>	Refers to the main AC power source typically provided to the wall AKA “mains”.
<i>Active Cable</i>	A cable with a <i>USB Type-C</i> plug on each end that incorporates data bus signal conditioning circuits. The cable supports the Structured VDM <i>Discover Identity Command</i> to expose its characteristics in addition to other Structured VDM <i>Commands</i> (Electronically Marked Cable see [USB Type-C 2.4]).
<i>Active Cable VDO</i>	VDO defining the Capabilities of an Active Cable.
<i>Active Mode</i>	A <i>Mode</i> which has been through the <i>Mode Entry</i> process but not the <i>Mode Exit</i> process.
<i>Adjustable Voltage Supply</i>	A power supply whose output voltage can be adjusted to an operating voltage within its Advertised range. These <i>Capabilities</i> are exposed by the <i>Adjustable Voltage Supply (AVS) APDO</i> (see Section 6.4.1.2.4, "Augmented Power Data Object (APDO)"). Note: Unlike the <i>SPR PPS</i> , the <i>SPR AVS</i> and <i>EPR AVS</i> do not support current limit.
<i>Advertise</i>	An offer made by a <i>Source</i> in the <i>Source_Capabilities/EPR_Source_Capabilities Message</i> (e.g., an <i>APDO</i> or <i>PDO</i>).
<i>Alternate Mode</i>	Operation defined by a Vendor or Standard’s organization, which is associated with a <i>SVID</i> . The definition of <i>Alternate Modes</i> is outside the scope of USB-IF specifications. Entry to and exit from the <i>Alternate Mode</i> uses the <i>Mode Entry</i> and <i>Mode Exit</i> processes. As defined in [USB Type-C 2.4] .
<i>Alternate Mode Adapter</i>	A <i>PDUSB Device</i> which supports <i>Alternate Modes</i> as defined in [USB Type-C 2.4] . Note: Since an <i>AMA</i> is a <i>PDUSB Device</i> , it has a single <i>UFP</i> that is only addressable by <i>SOP Packets</i> .
<i>Alternate Mode Controller</i>	A <i>DFP</i> that supports connection to <i>AMAs</i> as defined in [USB Type-C 2.4] . A <i>DFP</i> that is an <i>AMC</i> can also be a <i>PDUSB Host</i> .
<i>AMA</i>	See <i>Alternate Mode Adapter</i> .
<i>AMC</i>	See <i>Alternate Mode Controller</i> .
<i>AMS</i>	See <i>Atomic Message Sequence</i> .
<i>APDO</i>	See <i>Augmented Power Data Object</i> .
<i>Assured Capacity Charger</i>	As defined in [USB Type-C 2.4] . This maps to a <i>Charger</i> with one or more <i>Guaranteed Capability Ports</i> .
<i>Assured Capacity Group</i>	As defined in [USB Type-C 2.4] . This maps to a group of <i>Guaranteed Capability Ports</i> .
<i>Atomic Message Sequence</i>	A fixed sequence of <i>Messages</i> as defined in Section 8.3.2, "Atomic Message Sequence Diagrams" typically starting and ending in one of the following states: <i>PE_SRC_Ready</i> , <i>PE_SNK_Ready</i> or <i>PE_CBL_Ready</i> . An <i>AMS</i> is <i>Non-interruptible</i> .
<i>Attach</i>	Mechanical joining of the <i>Port Pair</i> by a cable.
<i>Attached</i>	USB Power Delivery <i>Ports</i> which are mechanically joined with USB cable.
<i>Attachment</i>	See <i>Attach</i> .
<i>Augmented Power Data Object</i>	<i>Data Object</i> used to expose a <i>Source Port's</i> or <i>Sink Port's</i> power <i>Capabilities</i> as part of a <i>Source_Capabilities/EPR_Source_Capabilities</i> or <i>Sink_Capabilities/EPR_Sink_Capabilities Message</i> respectively. An <i>SPR PPS Data Object</i> , <i>SPR AVS Data Object</i> and <i>EPR AVS Data Object</i> are defined.
<i>AVS</i>	See <i>Adjustable Voltage Supply</i> .
<i>AVS Mode</i>	A power supply, currently operating as an <i>AVS</i> , is said to be operating in <i>AVS Mode</i> .
<i>Battery</i>	A power storage device residing behind a <i>Port</i> that can either be a <i>Source</i> or <i>Sink</i> of power.

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Battery Slot</i>	A physical location where a <i>Hot Swappable Battery</i> can be installed. A <i>Battery Slot</i> might or might not have a <i>Hot Swappable Battery</i> present in a <i>Battery Slot</i> at any given time.
<i>Battery Supply</i>	A power supply that directly applies the output of a <i>Battery</i> to <i>VBUS</i> . This is exposed by the <i>Battery Supply PDO</i> (see Section 6.4.1.2.3, "Battery Supply Power Data Object").
<i>BDO</i>	See <i>BIST Data Object</i> .
<i>BFSK</i>	See <i>Binary Frequency Shift Keying</i> .
<i>Bi-phase Mark Coding</i>	Modification of Manchester coding where each zero has one transition and a one has two transitions (see [IEC 60958-1]).
<i>Binary Frequency Shift Keying</i>	A <i>Signaling Scheme</i> now Deprecated in this specification. <i>BFSK</i> used a pair of discrete frequencies to transmit binary (0s and 1s) information over <i>VBUS</i> . See [USBPD 2.0] for further details.
<i>BIST</i>	Built-In Self-Test - Power Delivery testing mechanism for the <i>PHY Layer</i> .
<i>BIST Data Object</i>	<i>Data Object</i> used by <i>BIST Messages</i> .
<i>BIST Mode</i>	A <i>BIST</i> receiver or transmitter test mode enabled by a <i>BIST Message</i> .
<i>BIST Carrier Mode</i>	A <i>BIST Mode</i> in which the <i>PHY Layer</i> sends out a <i>BMC</i> encoded continuous string of alternating "1"s and "0"s.
<i>BIST Test Data Mode</i>	A <i>BIST Mode</i> in which the <i>PHY Layer</i> sends out a <i>GoodCRC Message</i> and then enters a test mode where it sends no further <i>Messages</i> , except <i>GoodCRC Messages</i> , in response to received <i>Messages</i> .
<i>BIST Shared Capacity Test Mode</i>	A <i>BIST Mode</i> applicable only to a <i>Shared Capacity Group</i> of <i>Ports</i> where the maximum <i>Source Capabilities</i> are always offered on every <i>Port</i> , regardless of the availability of shared power i.e., all shared power management is disabled.
<i>BMC</i>	See <i>Bi-phase Mark Coding</i> .
<i>Cable Capabilities</i>	<i>Capabilities</i> offered by a <i>Cable Plug</i> .
<i>Cable Discovered</i>	USB Power Delivery <i>Ports</i> that have exchanged a <i>Message</i> and a <i>GoodCRC Message</i> response with a <i>Cable Plug</i> or a <i>VPD</i> using the USB Power Delivery protocol so that both the <i>Port</i> and the <i>Cable Plug</i> know that each is <i>PD Capable</i> and which <i>Revision</i> they each support.
<i>Cable Discovery</i>	See <i>Cable Discovered</i> .
<i>Cable Plug</i>	Term used to describe a <i>PD Capable</i> element in a <i>Multi-Drop</i> system addressed by <i>SOP' Packets</i> / <i>SOP'' Packets</i> . Logically the <i>Cable Plug</i> is associated with a <i>USB Type-C</i> plug at one end of the cable. In a practical implementation, the electronics might reside anywhere in the cable.
<i>Cable Reset</i>	This is initiated by <i>Cable Reset Signaling</i> from the <i>DFP</i> . It restores the <i>Cable Plugs</i> to their default, power up condition and resets the <i>PD</i> communications engine in the cable to its default state. It does not reset the <i>Port Partners</i> but does restore <i>VCONN</i> to its <i>Attachment</i> state.
<i>Cable VDO</i>	<i>VDO</i> returned by the <i>Cable Plug</i> containing <i>Cable Capabilities</i> .
<i>Capabilities</i>	Features supported by a product. These can include, for example, power levels supplied/needed, cable type, <i>Battery</i> support or [USB4] support.
<i>Capabilities Mismatch</i>	Indication from the <i>Sink</i> that the <i>Source's Advertised Capabilities</i> don't match the <i>Sink's</i> needs.
<i>CC</i>	See <i>Configuration Channel</i> .
<i>Cert Stat VDO</i>	The <i>Cert Stat VDO</i> contains the <i>XID</i> assigned by USB-IF to the product before certification in binary format.
<i>Charge Through</i>	A mechanism for a <i>VCONN Powered USB Device (VPD)</i> to pass power and <i>CC</i> communication from one <i>Port</i> to the other without any interference or re-regulation.
<i>Charge Through Port</i>	The <i>USB Type-C</i> receptacle on a <i>USB Device</i> that is designed to allow a <i>Source</i> to be connected through the <i>USB Device</i> to charge a system to which it is <i>Attached</i> . Most common use is to allow a single <i>Port USB Host</i> to support a <i>USB Device</i> while being charged.
<i>Charger</i>	<i>Provider</i> whose primary purpose is to supply power to a <i>Consumer</i> or <i>Consumers</i> in order to charge their <i>Battery</i> .
<i>Chunk</i>	A <i>MaxExtendedMsgChunkLen</i> (26 byte) or less portion of a <i>Data Block</i> . <i>Data Blocks</i> can be sent either as a single <i>Message</i> or as a series of <i>Chunks</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Chunked</i>	See <i>Chunking</i> .
<i>Chunked Extended Message</i>	<i>Extended Message</i> which has been broken up into <i>Chunks</i> .
<i>Chunking</i>	The process of breaking up a <i>Data Block</i> larger than <i>MaxExtendedMsgLegacyLen</i> (26-bytes) into two or more <i>Chunks</i> .
<i>Chunking Layer</i>	Part of the <i>Protocol Layer</i> responsible for <i>Chunking</i> .
<i>CL</i>	See <i>Current Limit</i> .
<i>Cold Socket</i>	A <i>Port</i> that does not apply <i>vSafe5V</i> on <i>VBUS</i> until a <i>Sink</i> is <i>Attached</i> .
<i>Collision Avoidance</i>	Mechanisms to prevent simultaneous communication by the <i>Source</i> , <i>Sink</i> and <i>Cable Plug</i> on <i>CC</i> .
<i>Command</i>	Request and response pair defined as part of a <i>Structured Vendor Defined Message</i> (see Section 6.4.4.2, "Structured VDM").
<i>Configuration Channel</i>	Single wire used by the <i>BMC PHY Layer Signaling Scheme</i> (see [USB Type-C 2.4]).
<i>Connect</i>	See <i>Connected</i> .
<i>Connected</i>	USB Power Delivery ports that have exchanged a <i>Message</i> and a <i>GoodCRC</i> <i>Message</i> response using the USB Power Delivery protocol so that both <i>Port Partners</i> know that each is <i>PD Capable</i> .
<i>Constant Voltage</i>	A constant voltage feature of an <i>SPR PPS Source</i> . The <i>SPR PPS Source</i> output voltage remains constant as the load changes up to its <i>Current Limit</i> .
<i>Consumer</i>	The capability of a <i>PD Port</i> (typically a <i>Device's UFP</i>) to sink power from the power conductor (e.g., <i>VBUS</i>). This corresponds to a <i>USB Type-C Port</i> with <i>R_d</i> asserted on its <i>CC</i> wire.
<i>Consumer/Provider</i>	A <i>Consumer</i> with the additional capability to function as a <i>Provider</i> . This corresponds to a <i>Dual-Role Power Port</i> with <i>R_d</i> asserted on its <i>CC</i> wire.
<i>Continuous BIST Mode</i>	The <i>BIST Mode</i> where the <i>Port</i> or <i>Cable Plug</i> being tested sends a continuous stream of test data.
<i>Contract</i>	An agreement on both power level and direction is reached between a <i>Port Pair</i> . A <i>Contract</i> could be explicitly <i>Negotiated</i> between the <i>Port Pair</i> or could be an implicit power level defined by the current <i>State</i> . While operating in Power Delivery mode there will always be either an <i>Explicit Contract</i> or <i>Implicit Contract</i> in place. The <i>Contract</i> can only be altered in the case of a <i>Negotiation/Re-negotiation</i> , <i>Power Role Swap</i> , <i>Fast Role Swap</i> , <i>Hard Reset</i> , <i>Error Recovery</i> or failure of the <i>Source</i> .
<i>Control Message</i>	A <i>Control Message</i> is defined as a <i>Message</i> with the <i>Number of Data Objects</i> field in the <i>Message Header</i> is set to zero. The <i>Control Message</i> consists only of a <i>Message Header</i> and a <i>CRC</i> .
<i>CRC</i>	<i>CRC</i> stands for Cyclic Redundancy Check. It is an error-detecting code used to determine if a block of data has been corrupted.
<i>CT-VPD</i>	See <i>VCONN Powered USB Charge Through Device</i> .
<i>Current Limit</i>	A current limiting feature of an <i>SPR PPS Source</i> . When a <i>Sink</i> operating in <i>SPR PPS</i> mode attempts to draw more current from the <i>Source</i> than the requested <i>Current Limit</i> value, the <i>Source</i> reduces its output voltage so the current it supplies remains at or below the requested value. Note: <i>Current Limit</i> is not supported by <i>SPR AVS</i> and <i>EPR AVS Sources</i> .
<i>CV</i>	See <i>Constant Voltage</i> .
<i>Data Block</i>	An <i>Extended Message Payload</i> data unit. The size of each type of <i>Data Block</i> is specified as a series of bytes up to <i>MaxExtendedMsgLen</i> bytes in length. This is distinct from a <i>Data Object</i> used by a <i>Data Message</i> which is always a 32-bit object.
<i>Data Message</i>	A <i>Data Message</i> consists of a <i>Message Header</i> followed by one or more <i>Data Objects</i> . <i>Data Messages</i> are easily identifiable because the <i>Number of Data Objects</i> field in the <i>Message Header</i> is always a non-zero value.
<i>Data Object</i>	A <i>Data Message Payload</i> data unit. This 32-bit object contains information specific to different types of <i>Data Message</i> . For example <i>Power</i> , <i>Request</i> , <i>BIST</i> , and <i>Vendor Data Objects</i> are defined.
<i>Data Reset</i>	Process which resets USB Communication.
<i>Data Role</i>	A <i>Port Partner</i> will be in one of two <i>Data Roles</i> ; either <i>DFP (USB Host)</i> or <i>UFP (USB Device)</i> .
<i>Data Role Swap</i>	Process of exchanging the <i>Data Roles</i> between <i>Port Partners</i> .
<i>Dead Battery</i>	A device has a <i>Dead Battery</i> when the <i>Battery</i> in a device is unable to power its functions.

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Default Contract</i>	An agreement on current at 5V is reached between a <i>Port Pair</i> based on <i>USB Type-C</i> current ([USB Type-C 2.4]).
<i>Detach</i>	Mechanical unjoining of the <i>Port Pair</i> by removal of the cable.
<i>Detached</i>	USB Power Delivery <i>Ports</i> which are no longer mechanically joined with USB cable.
<i>Detaches</i>	See <i>Detach</i> .
<i>Device</i>	When lower cased (device), it refers to any USB product, either <i>USB Device</i> or <i>USB Host</i> . When in upper case refers to a <i>USB Device (Peripheral or Hub)</i> .
<i>Device Policy</i>	Policy applied across multiple <i>Ports</i> in a <i>Source</i> or <i>Sink</i> .
<i>Device Policy Manager</i>	Module running in a <i>Source</i> or <i>Sink</i> that applies <i>Device Policy</i> to each <i>Port</i> in the device, as <i>Local Policy</i> , via the <i>Policy Engine</i> .
<i>DFP</i>	See <i>Downstream Facing Port</i> .
<i>DFP VDO</i>	<i>VDO</i> returned by the <i>DFP</i> containing <i>Capabilities</i> .
<i>Differential Non-Linearity</i>	The difference between an ideal <i>LSB</i> step, and the real observable <i>LSB</i> step when the Power <i>Source</i> is operating in either <i>PPS</i> or <i>AVS</i> mode. A <i>DNL</i> of 0 indicates that the step is ideal. If <i>DNL</i> is positive the step is larger than the ideal <i>LSB</i> , and if it is negative then the step is smaller than ideal.
<i>Discovery Process</i>	<i>Command</i> sequence using <i>Structured Vendor Defined Messages</i> resulting in identification of the <i>Port Partner</i> and <i>Cable Plug</i> , and their supported <i>SVIDs</i> and <i>Alternate Modes</i> .
<i>DNL</i>	See <i>Differential Non-Linearity</i> .
<i>Downstream Facing Port</i>	Indicates the <i>Port's</i> position in the USB topology which typically corresponds to a <i>USB Host</i> root <i>Port</i> or <i>Hub</i> downstream <i>Port</i> as defined in [USB Type-C 2.4] . At connection, the <i>Port</i> defaults to operation as the <i>Source</i> and as a <i>USB Host</i> (when <i>USB Communication</i> is supported).
<i>DPM</i>	See <i>Device Policy Manager</i> .
<i>DRD</i>	See <i>Dual-Role Data</i> .
<i>DRP</i>	See <i>Dual-Role Power</i> .
<i>Dual-Role Data</i>	Capability of operating as either a <i>DFP</i> or <i>UFP</i> .
<i>Dual-Role Data Port</i>	A <i>Port</i> capable of operating as <i>DRD</i> .
<i>Dual-Role Power</i>	Capability of operating as either a <i>Source</i> or <i>Sink</i> .
<i>Dual-Role Power Device</i>	A product containing one or more <i>Dual-Role Power Ports</i> that can operate as either a <i>Source</i> or a <i>Sink</i> .
<i>Dual-Role Power Port</i>	A <i>Port</i> capable of operating as a <i>DRP</i> .
<i>EM</i>	See <i>Extended Message</i> .
<i>End of Packet</i>	<i>K-code</i> marker used to delineate the end of a <i>Packet</i> .
<i>EOP</i>	See <i>End of Packet</i> .
<i>EPR</i>	See <i>Extended Power Range</i> .
<i>EPR AVS</i>	A power supply operating in <i>EPR Mode</i> whose output voltage can be adjusted to an operating voltage within its <i>Advertised</i> range. Unlike <i>SPR PPS</i> it does not support current limit. The <i>AVS Capabilities</i> are exposed by the <i>Adjustable Voltage Supply APDO</i> (see Section 6.4.1.2.4, "Augmented Power Data Object (APDO)").
<i>EPR AVS Mode</i>	A <i>EPR Source</i> , currently operating in an <i>EPR AVS Contract</i> , is said to be operating in <i>EPR AVS Mode</i> .
<i>EPR Cable</i>	A cable which is rated to operate in both <i>SPR Mode</i> and <i>EPR Mode</i> .
<i>EPR Capabilities</i>	The <i>EPR Capabilities Messages</i> (<i>EPR_Source_Capabilities</i> and <i>EPR_Sink_Capabilities</i>) are <i>Extended Messages</i> with the first seven positions filled with the same <i>SPR (A)PDOs</i> returned by the <i>SPR Capabilities Messages</i> (<i>Source_Capabilities</i> and <i>Sink_Capabilities</i>) followed by the <i>EPR (A)PDOs</i> starting in the eighth position.
<i>EPR Capable</i>	A product which has the ability to operate in <i>EPR Mode</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>EPR Mode</i>	A Power Delivery mode of operation where maximum allowable voltage is 48V. The <i>Sink</i> complies to the requirements of [IEC 62368-1] for operation with a PS3 <i>Source</i> . The <i>Source</i> complies to the requirements of [IEC 62368-1] for operation with a PS3 <i>Sink</i> . The cable complies with [IEC 62368-1] . Entry into the <i>EPR Mode</i> requires that an <i>EPR Source</i> is <i>Attached</i> to an <i>EPR Sink</i> with an <i>EPR Cable</i> . The <i>EPR Source</i> will only enter the <i>EPR Mode</i> when requested to do so by the <i>Sink</i> and it has determined it is <i>Attached</i> to an <i>EPR Sink</i> with an <i>EPR Capable</i> cable. Only the <i>EPR_Source_Capabilities</i> and the <i>EPR_Request</i> Messages are allowed to <i>Negotiate EPR Explicit Contracts</i> . The <i>SPR Mode Messages</i> (<i>Source_Capabilities</i> and <i>Request</i>) are not allowed to be used while in <i>EPR Mode</i> .
<i>EPR (A)PDO</i>	<i>Fixed Supply PDO</i> that offers either 28V, 36V or 48V. <i>Adjustable Voltage Supply (AVS) APDO</i> whose Maximum voltage is the highest <i>Fixed Supply PDO</i> voltage in the <i>EPR_Source_Capabilities</i> Message and no more than 240W.
<i>EPR Sink</i>	A <i>Sink</i> that supports both <i>SPR Mode</i> and <i>EPR Mode</i> .
<i>EPR Sink Port</i>	A <i>Port</i> exposed on an <i>EPR Sink</i> .
<i>EPR Source</i>	A <i>Source</i> that supports both <i>SPR Mode</i> and <i>EPR Mode</i> .
<i>EPR Source Port</i>	A <i>Port</i> exposed on an <i>EPR Source</i> .
<i>Error Recovery</i>	<i>Port</i> enters the <i>ErrorRecovery</i> State as defined in [USB Type-C 2.4] .
<i>Explicit Contract</i>	An agreement reached between a <i>Port Pair</i> as a result of the Power Delivery <i>Negotiation</i> process. An <i>Explicit Contract</i> is established (or continued) when a <i>Source</i> sends an <i>Accept</i> Message in response to a <i>Request</i> Message sent by a <i>Sink</i> followed by a <i>PS_RDY</i> Message sent by the <i>Source</i> to indicate that the power supply is ready. This corresponds to the <i>PE_SRC_Ready</i> State for a <i>Source Policy Engine</i> and the <i>PE_SNK_Ready</i> State for a <i>Source Policy Engine</i> . The <i>Explicit Contract</i> can be altered through the <i>Re-negotiation</i> process.
<i>Extended Capabilities</i>	An <i>Extended Message</i> containing <i>Capabilities</i> information.
<i>Extended Control Message</i>	An <i>Extended Message</i> containing control information only.
<i>Extended Message</i>	A Message containing <i>Data Blocks</i> . The <i>Extended Message</i> is defined by the <i>Extended</i> field in the <i>Message Header</i> being set to one and contains an <i>Extended Message Header</i> immediately following the <i>Message Header</i> .
<i>Extended Message Header</i>	Every <i>Extended Message</i> contains a 16-bit <i>Extended Message Header</i> immediately following the <i>Message Header</i> containing information about the <i>Data Block</i> and any <i>Chunking</i> being applied.
<i>Extended Power Range</i>	Extends the power range from a maximum of 100W (<i>SPR</i>) to a maximum of 240W (<i>EPR</i>). When operating in the <i>EPR Mode</i> , only <i>EPR</i> specific Messages (the <i>EPR_Source_Capabilities</i> Message and the <i>EPR_Request</i> Message) are used to <i>Negotiate Explicit Contracts</i> .
<i>External Supply</i>	Power supply external to the device. This could be powered from the wall or from any other power source.
<i>Fast Role Swap</i>	Process of exchanging the <i>Source</i> and <i>Sink Power Roles</i> between <i>Port Partners</i> rapidly due to the disconnection of an external power supply.
<i>Fast Role Swap Request</i>	An indication from an <i>Initial Source</i> to the <i>Initial Sink</i> that a <i>Fast Role Swap</i> is needed. The <i>Fast Role Swap Request</i> is indicated by driving the <i>CC</i> line to ground for a short period; it is not a <i>Message</i> or <i>Signaling</i> .
<i>First Explicit Contract</i>	The <i>Explicit Contract</i> that immediately follows an <i>Attach</i> , power on <i>Hard Reset</i> , <i>Power Role Swap</i> or <i>Fast Role Swap</i> event.
<i>Fixed Battery</i> <i>Fixed Batteries</i>	A <i>Battery</i> that is not easily removed or replaced by an end user e.g., requires a special tool to access or is soldered in.
<i>Fixed Supply</i>	A well-regulated fixed voltage power supply. This is exposed by the <i>Fixed Supply PDO</i> (see Section 6.4.1.2.1, "Fixed Supply Power Data Object")
<i>Frame</i>	Generic term referring to an atomic communication transmitted by <i>PD</i> such as a <i>Packet</i> , <i>Test Frame</i> or <i>Signaling</i> .
<i>FRS</i>	See <i>Fast Role Swap</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Guaranteed Capability Port</i>	A <i>Guaranteed Capability Port</i> is always capable of delivering its Port Maximum PDP and indicates this by setting its Port Present PDP to be the same as its Port Maximum PDP except when limited by the cable's <i>Capabilities</i> . This is a Static capability.
<i>Hard Reset</i>	This is initiated by Hard Reset Signaling from either <i>Port Partner</i> . It restores <i>VBUS</i> to <i>USB Default Operation</i> and resets the <i>PD</i> communications engine to its default <i>State</i> in both <i>Port Partners</i> as well as in any <i>Attached Cable Plugs</i> . It restores both <i>Port Partners</i> to their default <i>Data Roles</i> and returns the <i>VCONN Source</i> to the <i>Source Port</i> . A <i>DRP Source Port</i> operating as a <i>Source</i> will continue to operate as a <i>Source</i> .
<i>Host</i>	See <i>USB Host</i> .
<i>Hot Swappable Battery</i>	A <i>Battery</i> that is easily accessible for a user to remove or change for another <i>Battery</i> .
<i>Hub</i>	A <i>USB Device</i> that provides additional connections to the <i>USB</i> .
<i>ID Header VDO</i>	The <i>VDO</i> in a Discover Identity Command immediately following the <i>VDM Header</i> . The <i>ID Header VDO</i> contains information corresponding to the <i>Power Delivery Product</i> .
<i>Idle</i>	Condition on <i>CC</i> where there are no signal transitions within a given time window. See Section 5.8.6.1, "Definition of Idle" .
<i>Implicit Contract</i>	An agreement on power levels between a <i>Port Pair</i> which occurs, not because of the <i>Power Delivery Negotiation</i> process, but because of a <i>Power Role Swap</i> or <i>Fast Role Swap</i> . <i>Implicit Contracts</i> are transitory since the <i>Port Pair</i> is required to immediately <i>Negotiate</i> an <i>Explicit Contract</i> after the <i>Power Role Swap</i> . An <i>Implicit Contract</i> shall be limited to <i>USB Type-C</i> current (see [USB Type-C 2.4]).
<i>Initial Sink</i>	<i>Sink</i> at the start of a <i>Power Role Swap</i> or <i>Fast Role Swap</i> which transitions to being the <i>New Source</i> .
<i>Initial Source</i>	<i>Source</i> at the start of a <i>Power Role Swap</i> or <i>Fast Role Swap</i> which transitions to being the <i>New Sink</i> .
<i>Initiator</i>	The initial sender of a <i>Command</i> request in the form of a query.
<i>Invariant PDOs</i>	A <i>Source Port</i> that offers <i>Invariant PDOs</i> will always <i>Advertise</i> the same <i>PDOs</i> except when limited by the cable.
<i>IoC</i>	The <i>Negotiated</i> current value as defined in [IEC 63002] .
<i>IR Drop</i>	The voltage drop across the cable and connectors between the <i>Source</i> and the <i>Sink</i> as defined in [USB Type-C 2.4] . It is a function of the resistance of the ground and power wire in the cable plus the contact resistance in the connectors times the current flowing over the path.
<i>K-code</i>	Special symbols provided by the 4b5b coding scheme. <i>K-codes</i> are used to signal <i>Hard Reset</i> and <i>Cable Reset</i> and delineate <i>Packet</i> boundaries.
<i>Local Policy</i>	Every <i>PD Capable</i> device has its own <i>Policy</i> , called the <i>Local Policy</i> that is executed by its <i>Policy Engine</i> to control its power delivery behavior. The <i>Local Policy</i> at any given time might be the default policy, hard coded or modified by changes in operating parameters or one provided by the system <i>USB Host</i> or some combination of these. The <i>Local Policy</i> Optionally can be changed by a <i>System Policy Manager</i> .
<i>LPS</i>	Limited Power Supply as defined in [IEC 62368-1] .
<i>LSB</i>	An abbreviation for Least Significant Bit.
<i>Managed Capability Port</i>	A <i>Managed Capability Port</i> can have its Port Present PDP set to a different value than its Port Maximum PDP . Its Port Present PDP value can be dynamic and change during normal operation.
<i>Message</i>	The <i>Packet Payload</i> consisting of a <i>Message Header</i> for <i>Control Messages</i> and a <i>Message Header</i> and data for <i>Data Messages</i> and <i>Extended Messages</i> as defined in Section 6.2, "Messages" .
<i>Message Header</i>	Every <i>Message</i> starts with a 16-bit <i>Message Header</i> containing basic information about the <i>Message</i> and the <i>PD Port's Capabilities</i> .
<i>Messaging</i>	Communication in the form of <i>Messages</i> as defined in Section 6, "Protocol Layer" .
<i>Modal Operation</i>	Operation where there are one or more <i>Active Modes</i> . <i>Modal Operation</i> ends when there are no longer any <i>Active Modes</i> .
<i>Mode</i>	<i>Mode</i> is a general term used to describe a particular type of operation of a given device. Examples of modes are: <i>Alternate Mode</i> , <i>EPR Mode</i> , <i>SPR Mode</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Mode Entry</i>	Process to start operation in a particular <i>Mode</i> .
<i>Mode Exit</i>	Process to end operation in a particular <i>Mode</i> .
<i>Multi-Drop</i>	PD is a <i>Multi-Drop</i> system sharing the Power Delivery communication channel between the <i>Port Partners</i> and the cable.
<i>Negotiate</i>	See <i>Negotiation</i> .
<i>Negotiated</i>	See <i>Negotiation</i> .
<i>Negotiation</i>	<p>This is the PD process whereby:</p> <ol style="list-style-type: none"> 1) The <i>Source</i> Advertises its <i>Capabilities</i>. 2) The <i>Sink</i> requests one of the <i>Advertised Capabilities</i>. 3) The <i>Source</i> acknowledges the request, alters its output to satisfy the request and informs the <i>Sink</i>. <p>The result of the <i>Negotiation</i> is a <i>Contract</i> for power delivery/consumption between the <i>Port Pair</i>.</p>
<i>New Sink</i>	<i>Sink</i> at the end of a <i>Power Role Swap</i> or <i>Fast Role Swap</i> which has transition from being the <i>Initial Source</i> .
<i>New Source</i>	<i>Source</i> at the end of a <i>Power Role Swap</i> or <i>Fast Role Swap</i> which has transition from being the <i>Initial Sink</i> .
<i>Non-interruptible</i>	There cannot be any unexpected <i>Messages</i> during an <i>AMS</i> ; it is therefore <i>Non-interruptible</i> . An <i>AMS</i> starts when the first <i>Message</i> in the <i>AMS</i> has been sent (i.e., a <i>GoodCRC</i> <i>Message</i> has been received acknowledging the <i>Message</i>). See Section 8.3.2.1.3, "Atomic Message Sequences" .
<i>OCP</i>	Over-Current Protection.
<i>OTP</i>	Over-Temperature Protection.
<i>OVP</i>	Over-Voltage Protection.
<i>Packet</i>	One entire unit of PD communication including a <i>Preamble</i> , <i>SOP*</i> , <i>Payload</i> , <i>CRC</i> and <i>EOP</i> as defined in Section 5.6, "Packet Format" .
<i>Passive Cable</i>	<p>Cable with a USB plug on each end at least one of which is a <i>Cable Plug</i> supporting <i>SOP'</i> that does not incorporate data bus signal conditioning circuits. Supports the <i>Structured VDM Discover Identity</i> to determine its characteristics (Electronically Marked Cable see [USB Type-C 2.4]).</p> <p>Note: This specification does not discuss <i>Passive Cables</i> that are not Electronically Marked.</p>
<i>Passive Cable VDO</i>	VDO defining the <i>Capabilities</i> of a <i>Passive Cable</i> .
<i>Payload</i>	Data content of a <i>Packet</i> , provided to/from the <i>Protocol Layer</i> .
<i>PD</i>	USB Power Delivery
<i>PD Capable</i>	A <i>Port</i> that supports USB Power Delivery.
<i>PD Connection</i>	See <i>Connected</i> .
<i>PD Power</i>	The output power, in Watts, of a <i>Source</i> , as specified by the manufacturer and expressed in <i>Fixed Supply PDOs</i> as defined in Section 10, "Power Rules" .
<i>PD SID</i>	See <i>USB-IF PD SID</i> .
<i>PDO</i>	See <i>Power Data Object</i> .
<i>PDP</i>	See <i>PD Power</i> .
<i>PDP Rating</i>	The <i>PDP Rating</i> is the same as the Manufacturer declared <i>PDP</i> for a <i>Source Port</i> except where there is a fractional value, in which case the <i>PDP Rating</i> corresponds to the integer part of the Manufacturer declared <i>PDP Rating</i> (see Section 6.4.11.2, "Port Maximum PDP Field").
<i>PDUSB</i>	<i>USB Device Port</i> or <i>USB Host Port</i> that is both <i>PD Capable</i> and capable of <i>USB Communication</i> . See also <i>PDUSB Host</i> , <i>PDUSB Device</i> and <i>PDUSB Hub</i> .
<i>PDUSB Device</i>	A <i>USB Device</i> with a <i>PD Capable UFP</i> . A <i>PDUSB Device</i> is only addressed by <i>SOP Packets</i> .
<i>PDUSB Host</i>	A <i>USB Host</i> which is <i>PD Capable</i> on at least one of its <i>DFPs</i> . A <i>PDUSB Host</i> is only addressed by <i>SOP Packets</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>PDUSB Hub</i>	A port expander <i>USB Device</i> with a <i>UFP</i> and one or more <i>DFPs</i> which is <i>PD Capable</i> on at least one of its <i>Ports</i> . A <i>PDUSB Hub</i> is only addressed by <i>SOP Packets</i> . A self-powered <i>PDUSB Hub</i> is treated as a <i>USB Type-C Multi-Port Charger</i> .
<i>PDUSB Peripheral</i>	A <i>USB Device</i> with a <i>PD Capable UFP</i> which is not a <i>PDUSB Hub</i> . A <i>PDUSB Peripheral</i> is only addressed by <i>SOP Packets</i> .
<i>PE</i>	See <i>Policy Engine</i> .
<i>Peripheral</i>	A physical entity that is <i>Attached</i> to a <i>USB cable</i> and is currently operating as a <i>USB Device</i> .
<i>PHY Layer</i>	The Physical Layer responsible for sending and receiving <i>Messages</i> across the <i>USB Type-C CC</i> wire between a <i>Port Pair</i> .
<i>Policy</i>	<i>Policy</i> defines the behavior of <i>PD Capable</i> parts of the system and defines the <i>Capabilities</i> it <i>Advertises</i> , requests made to (re) <i>Negotiate</i> power and the responses made to requests received.
<i>Policy Engine</i>	The <i>Policy Engine</i> interprets the <i>Device Policy Manager's</i> input to implement <i>Policy</i> for a given <i>Port</i> and directs the <i>Protocol Layer</i> to send appropriate <i>Messages</i> .
<i>Port</i>	An interface typically exposed through a receptacle, or via a plug on the end of a hard-wired captive cable. <i>USB Power Delivery</i> defines the interaction between a <i>Port Pair</i> .
<i>Port Pair</i>	Two <i>Attached PD Capable Ports</i> .
<i>Port Partner</i>	A <i>Contract</i> is <i>Negotiated</i> between a <i>Port Pair</i> connected by a <i>USB cable</i> . These ports are known as <i>Port Partners</i> .
<i>Power Conductor</i>	The wire that delivers power from the <i>Source</i> to <i>Sink</i> . For example, <i>USB's VBUS</i> .
<i>Power Consumer</i>	See <i>Consumer</i> .
<i>Power Data Object</i>	<i>Data Object</i> used to expose a <i>Source Port's</i> or <i>Sink Port's</i> power <i>Capabilities</i> as part of a <i>Source_Capabilities</i> / <i>EPR_Source_Capabilities</i> or <i>Sink_Capabilities</i> / <i>EPR_Sink_Capabilities</i> <i>Message</i> respectively. <i>Fixed Supply</i> , <i>Variable Supply</i> and <i>Battery Supply</i> <i>Power Data Objects</i> are defined; <i>SPR Mode</i> uses all four while <i>EPR Mode</i> uses only <i>Fixed Supply</i> and <i>AVS</i> <i>PDOs</i> .
<i>Power Delivery Mode</i>	Operation after a <i>Contract</i> has initially been established between a <i>Port Pair</i> . This <i>Mode</i> persists during normal <i>Power Delivery</i> operation, including after a <i>Power Delivery Mode</i> . <i>Power Delivery Mode</i> can only be exited by <i>Detaching</i> the <i>Ports</i> , applying a <i>Hard Reset</i> or by the <i>Source</i> removing power (except when the <i>Initial Source</i> removes power from <i>VBUS</i> during the <i>Power Role Swap</i> procedure).
<i>Power Provider</i>	See <i>Provider</i> .
<i>Power Role</i>	A <i>Port Partner</i> will be in one of two <i>Power Roles</i> ; either <i>Source</i> or <i>Sink</i> .
<i>Power Role Swap</i>	Process of exchanging the <i>Source</i> and <i>Sink Power Roles</i> between <i>Port Partners</i> .
<i>Power Rules</i>	Define voltages and current ranges that are offered by compliant <i>USB Power Delivery Sources</i> and used by a <i>USB Power Delivery Sink</i> for a given value of <i>PDP Rating</i> . See Section 10, "Power Rules" .
<i>PPS</i>	See <i>Programmable Power Supply</i> .
<i>PPS Mode</i>	An <i>SPR Source</i> , currently operating as an <i>PPS</i> , is said to be operating in <i>PPS Mode</i> .
<i>Preamble</i>	Start of a transmission which is used to enable the receiver to lock onto the carrier. The <i>Preamble</i> consists of a 64-bit sequence of alternating 0s and 1s starting with a "0" and ending with a "1" which is not 4b5b encoded.
<i>Product Type</i>	Product categorization returned as part of the <i>Discover Identity</i> <i>Command</i> .
<i>Product Type VDO</i>	<i>VDO</i> identifying a certain <i>Product Type</i> in the <i>ID Header VDO</i> of a <i>Discover Identity</i> <i>Command</i> .
<i>Product VDO</i>	The <i>Product VDO</i> contains identity information relating to the product.
<i>Programmable Power Supply</i>	A power supply, operating in <i>SPR Mode</i> , whose output voltage can be programmatically adjusted in small increments over its <i>Advertised</i> range and has a programmable output current fold back (note that the <i>SPR AVS</i> and <i>EPR AVS</i> does not). The <i>Capabilities</i> are exposed by the <i>SPR Programmable Power Supply APDO</i> (see Section 6.4.1.2.4, "Augmented Power Data Object (APDO)").
<i>Protocol Error</i>	An unexpected <i>Message</i> during an <i>Atomic Message Sequence</i> . A <i>Protocol Error</i> during an <i>AMS</i> will result in either a <i>Soft Reset</i> or a <i>Hard Reset</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Protocol Layer</i>	The entity that forms the <i>Messages</i> used to communicate information between <i>Port Partners</i> .
<i>Provider</i>	A <i>PD Port</i> (typically a <i>USB Host</i> , <i>Hub</i> , or <i>Charger DFP</i>) that can source power over the power conductor (e.g., <i>V_{BUS}</i>). This corresponds to a <i>USB Type-C Port</i> with R_p asserted on its <i>CC</i> wire.
<i>Provider/Consumer</i>	A <i>Provider</i> with the additional capability to act as a <i>Consumer</i> . This corresponds to a <i>Dual-Role Power Port</i> with R_p asserted on its <i>CC</i> wire.
<i>PS1</i> <i>PS2</i> <i>PS3</i>	Classification of electrical power as defined in [IEC 62368-1] .
<i>PSD</i>	<i>Sink</i> which draws power but has no other <i>USB</i> or <i>Alternate Mode</i> communication function e.g., a power bank.
R_a	Prior to application of <i>V_{CONN}</i> , a powered cable applies a pull-down resistor R_a on its <i>V_{CONN}</i> pin.
R_d	Pull-down resistor on the <i>USB Type-C CC</i> wire used to indicate that the <i>Port</i> is a <i>Sink</i> (see [USB Type-C 2.4]).
<i>RDO</i>	See <i>Request Data Object</i> .
<i>Re-attach</i>	<i>Attach</i> of the <i>Port Pair</i> by a cable after a previous <i>Detach</i> .
<i>Re-negotiate</i>	See <i>Re-negotiation</i> .
<i>Re-negotiated</i>	See <i>Re-negotiation</i> .
<i>Re-negotiation</i>	A process wherein one of the <i>Port Partners</i> wants to alter the <i>Negotiated Contract</i> .
<i>Request</i>	<i>Message</i> used by a <i>Sink Port</i> to <i>Negotiate</i> a <i>Contract</i> ; refers to either a Request/EPR_Request Message .
<i>Request Data Object</i>	<i>Data Object</i> used by a <i>Sink Port</i> to <i>Negotiate</i> a <i>Contract</i> as a part of a Request/EPR_Request Message .
<i>Responder</i>	The receiver of a <i>Command</i> request sent by an <i>Initiator</i> that replies with a <i>Command</i> response.
<i>Revision</i>	Major release of the <i>USB Power Delivery</i> specification. Each <i>Revision</i> will have various <i>Versions</i> associated with it.
<i>Revision 1.0</i>	Deprecated major <i>Revision</i> of the <i>USB Power Delivery</i> Specification.
<i>Revision 2.0</i>	Superseded major <i>Revision</i> of the <i>USB Power Delivery</i> Specification as defined in [USBPD 2.0] , with which this specification is compatible.
<i>Revision 3.x</i>	Current major <i>Revisions</i> of the <i>USB Power Delivery</i> Specification.
R_p	Pull-up resistor on the <i>USB Type-C CC</i> wire used to indicate that the <i>Port</i> is a <i>Source</i> (see [USB Type-C 2.4]).
<i>Safe Operation</i>	<i>Sources</i> must have the ability to tolerate vSafe5V applied by both <i>Port Partners</i> .
<i>Shared Capacity Charger</i>	As defined in [USB Type-C 2.4] . This maps to a <i>Charger</i> with multiple <i>Managed Capability Ports</i> .
<i>Shared Capacity Group</i>	As defined in [USB Type-C 2.4] . This maps to a group with <i>Managed Capability Ports</i> .
<i>SID</i>	See <i>Standard ID</i> .
<i>Signaling</i>	A Preamble followed by an ordered set of four <i>K-codes</i> used to indicate a particular line symbol e.g., <i>Hard Reset</i> as defined in Section 5.4, "Ordered Sets" .
<i>Signaling Scheme</i>	Physical mechanism used to transmit bits. Only the <i>BMC Signaling Scheme</i> is defined in this specification. Note: The <i>BFSK Signaling Scheme</i> supported in <i>Revision 1.0</i> of this specification has been Deprecated .
<i>Single-Role Port</i>	A <i>Port</i> that is only capable of operating either as a <i>Source</i> or <i>Sink</i> , but not both. E.g., the port is not a <i>DRP</i> .
<i>Sink</i>	The <i>Port</i> consuming power from <i>V_{BUS}</i> ; most commonly a <i>USB Device</i> .
<i>Sink Capabilities</i>	<i>Capabilities</i> wanted by a <i>Sink</i> .
<i>Sink Directed Charge</i>	A charging scheme whereby the <i>Sink</i> connects the <i>Source</i> to its <i>Battery</i> through safety and other circuitry. When the <i>SPR PPS Current Limit</i> feature is activated, the <i>Source</i> automatically controls its output current by adjusting its output voltage.

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Sink Port</i>	<i>Port</i> operating as a <i>Sink</i> .
<i>Sink Standby</i>	During <i>Sink Standby</i> the <i>Sink</i> reduces its current draw to <i>iSnkStdby</i> .
<i>Soft Reset</i>	A process that resets the <i>PD</i> communications engine to its default state.
<i>SOP</i>	<i>K-code</i> marker used for communication between <i>Port Partners</i> . See also <i>Start of Packet</i> .
<i>SOP Communication</i>	Communication using <i>SOP Packets</i> also implies that an <i>AMS</i> is being followed.
<i>SOP Packet</i>	Any Power Delivery <i>Packet</i> which starts with an <i>SOP</i> .
<i>SOP' Communication</i>	Communication with a <i>Cable Plug</i> using <i>SOP' Packets</i> , also implies that an <i>AMS</i> is being followed.
<i>SOP' Packet</i>	Any Power Delivery <i>Packet</i> which starts with an <i>SOP'</i> used to communicate with a <i>Cable Plug</i> .
<i>SOP'' Communication</i>	Communication with a <i>Cable Plug</i> using <i>SOP'' Packets</i> , also implies that an <i>AMS</i> is being followed.
<i>SOP'' Packet</i>	Any Power Delivery <i>Packet</i> which starts with an <i>SOP''</i> used to communicate with a <i>Cable Plug</i> when <i>SOP' Packets</i> are being used to communicate with the other <i>Cable Plug</i> .
<i>SOP'</i> <i>SOP''</i>	<i>K-code</i> marker used for communication between a <i>Port</i> and a <i>Cable Plug</i> . See also <i>Start of Packet</i> .
<i>SOP*</i>	Used to generically refer to <i>K-code</i> markers: <i>SOP</i> , <i>SOP'</i> and <i>SOP''</i> . See also <i>Start of Packet</i> .
<i>SOP* Communication</i>	Communication using <i>SOP* Packets</i> , also implies an <i>AMS</i> is being followed.
<i>SOP* Packet</i>	A term referring to any Power Delivery <i>Packet</i> starting with either <i>SOP</i> , <i>SOP'</i> , or <i>SOP''</i> .
<i>Source</i>	The Power Role a <i>Port</i> is operating in to supply power over <i>VBUS</i> ; most commonly a <i>USB Host</i> or <i>Hub</i> downstream port.
<i>Source Capabilities</i>	<i>Capabilities</i> offered by a <i>Source</i> .
<i>Source Port</i>	<i>Port</i> operating as a <i>Source</i> .
<i>Specification Revision</i>	See <i>Revision</i> .
<i>SPM</i>	See <i>System Policy Manager</i> .
<i>SPR</i>	See <i>Standard Power Range</i> .
<i>SPR AVS</i>	An <i>SPR Source</i> whose output voltage can be adjusted to an operating voltage within its <i>Advertised</i> range. Unlike <i>SPR PPS</i> , it does not support current limit. The <i>SPR AVS Capabilities</i> are exposed by the <i>SPR AVS APDO</i> (see Section 6.4.1.2.4.2, "SPR Adjustable Voltage Supply APDO").
<i>SPR AVS Mode</i>	A <i>SPR Source</i> , currently operating in an <i>SPR AVS Contract</i> , is said to be operating in <i>SPR AVS Mode</i> .
<i>SPR Capabilities</i>	An <i>SPR Capabilities Message</i> (<i>Source Capabilities Message</i> or <i>Sink Capabilities Message</i>) has at least one <i>Power Data Object</i> for <i>vSafe5V</i> followed by up to 6 additional <i>Power Data Objects</i> .
<i>SPR Contract</i>	<i>Explicit Contract Negotiated</i> , in <i>SPR Mode</i> , based on <i>SPR (A)PDOs</i> .
<i>SPR Mode</i>	The classic mode of <i>PD</i> operation where <i>Explicit Contracts</i> are <i>Negotiated</i> using <i>SPR (A)PDOs</i> .
<i>SPR (A)PDO</i>	<i>Fixed Supply PDO</i> that offers up to 20V and no more than 100W. <i>Variable Supply PDO</i> whose Maximum voltage offers up to 21V and no more than 100W. <i>Battery Supply PDO</i> whose Maximum voltage offers up to 21V and no more than 100W. <i>Adjustable Voltage Supply (AVS) APDO</i> whose Maximum voltage is up to 20V and no more than 100W. <i>Programmable Power Supply (PPS) APDO</i> whose Maximum voltage is up to 21V and no more than 100W.
<i>SPR PPS</i>	A power supply whose output voltage and output current can be programmatically adjusted in small increments over its <i>Advertised</i> range. It supports current limit unlike <i>SPR AVS</i> and <i>EPR AVS</i> . The <i>Capabilities</i> are exposed by the <i>Programmable Power Supply APDOs</i> (see Section 6.4.1.2.4, "Augmented Power Data Object (APDO)").
<i>SPR PPS Mode</i>	A power supply, currently operating in an <i>SPR PPS Contract</i> , is said to be operating in <i>SPR PPS Mode</i> .
<i>SPR Sink</i>	A <i>Sink</i> which only supports <i>SPR Mode</i> and does not support <i>EPR Mode</i> .
<i>SPR Sink Port</i>	A <i>Port</i> exposed on an <i>SPR Sink</i> .
<i>SPR Source</i>	A <i>Source</i> which only supports <i>SPR Mode</i> and does not support <i>EPR Mode</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>SPR Source Port</i>	A <i>Port</i> exposed on an <i>SPR Source</i> .
<i>Standard ID</i>	16-bit unsigned value assigned by the USB-IF to a given industry standards organization's specification.
<i>Standard or Vendor ID</i>	Generic term referring to either a <i>VID</i> or a <i>SID</i> . <i>SVID</i> is used in place of the phrase "Standard or Vendor ID."
<i>Standard Power Range</i>	Only the <i>Source Capabilities</i> and the <i>Request Messages</i> are allowed to <i>Negotiate SPR Explicit Contracts</i> . The <i>EPR Messages</i> (the <i>EPR_Source Capabilities Message</i> and the <i>EPR_Request Message</i>) are not allowed to be used while in <i>SPR Mode</i> .
<i>Start of Packet</i>	<i>K-code</i> marker used to delineate the start of a <i>Packet</i> .
<i>State</i>	<i>PD</i> state machine state as defined in Section 6.12, "State behavior" and Section 8.3.3, "State Diagrams" state machines.
<i>Structured VDM</i>	See <i>Structured Vendor Defined Message</i> .
<i>Structured VDM Header</i>	The <i>VDM Header</i> for a <i>Structured Vendor Defined Message</i> .
<i>Structured Vendor Defined Message</i>	A <i>Vendor Defined Message</i> where the contents and usage of bits 14...0 of the <i>VDM Header</i> are defined by this specification.
<i>SVDM</i>	See <i>Structured Vendor Defined Message</i> .
<i>SVID</i>	See <i>Standard or Vendor ID</i> .
<i>Swap Standby</i>	During <i>Swap Standby</i> the <i>Source</i> does not drive <i>VBUS</i> and the Sink's current draw does not exceed <i>iSnkSwapStdby</i> .
<i>System Policy</i>	Overall system <i>Policy</i> generated by the system, broken up into the policies required by each <i>Port Pair</i> to affect the <i>System Policy</i> . It is programmatically fed to the individual devices for consumption by their <i>Policy Engines</i> .
<i>System Policy Manager</i>	Module running on the <i>USB Host</i> . It applies the <i>System Policy</i> through communication with <i>PD Capable Consumers</i> and <i>Providers</i> that are also connected to the <i>USB Host</i> via <i>USB</i> .
<i>Test Frame</i>	<i>Frame</i> consisting of a <i>Preamble</i> , <i>SOP*</i> , followed by test data (See Section 5.9, "Built in Self-Test (BIST)").
<i>Test Pattern</i>	Continuous stream of test data in a given sequence (See Section 5.9, "Built in Self-Test (BIST)").
<i>Tester</i>	The <i>Tester</i> is assumed to be a piece of test equipment that manages the <i>BIST</i> testing process of a <i>PD UUT</i> .
<i>UFP</i>	See <i>Upstream Facing Port</i> .
<i>UFP VDO</i>	<i>VDO</i> returned by the <i>UFP</i> containing <i>Capabilities</i> .
<i>UI</i>	See <i>Unit Interval</i> .
<i>Unchunked</i>	See <i>Unchunked Extended Message</i> .
<i>Unchunked Extended Message</i>	<i>Extended Message</i> that has been transmitted whole without using <i>Chunking</i> .
<i>Unexpected Message</i>	<i>Message</i> that a <i>Port</i> supports but has been received in an incorrect <i>State</i> .
<i>Unit Interval</i>	The time to transmit a single data bit on the wire.
<i>Unit Under Test</i>	The <i>PD</i> device that is being tested by the <i>Tester</i> and responds to the initiation of a particular <i>BIST</i> test sequence.
<i>Unrecognized Message</i>	<i>Message</i> that a <i>Port</i> does not understand e.g., a <i>Message</i> using a Reserved <i>Message</i> type, a <i>Message</i> defined by a higher specification <i>Revision</i> than the <i>Revision</i> this <i>Port</i> supports, or an <i>Unstructured Vendor Defined Message</i> for which the <i>VID</i> is not recognized.
<i>Unstructured VDM</i>	See <i>Unstructured Vendor Defined Message</i> .
<i>Unstructured VDM Header</i>	The <i>VDM Header</i> for an <i>Unstructured Vendor Defined Message</i> .
<i>Unstructured Vendor Defined Message</i>	A <i>Vendor Defined Message</i> where the contents of bits 14...0 of the <i>VDM Header</i> are undefined.
<i>Unsupported Message</i>	<i>Message</i> that a <i>Port</i> recognizes but does not support. This is a <i>Message</i> defined by the specification, but which is not supported by this <i>Port</i> .

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>Upstream Facing Port</i>	Indicates the <i>Port</i> 's position in the USB topology typically a <i>Port</i> on a <i>Device</i> as defined in [USB Type-C 2.4] . At connection, the <i>Port</i> defaults to operation as a <i>USB Device</i> (when <i>USB Communication</i> is supported) and <i>Sink</i> .
<i>USB Attached State</i>	Synonymous with the [USB 2.0] and [USB 3.2] definition of the <i>Attached</i> state
<i>USB Communication</i>	Transfer of USB data <i>Packets</i> as defined in [USB 2.0] and [USB 3.2] .
<i>USB Default Operation</i>	Operation of a <i>Port</i> at <i>Attach</i> or after a <i>Hard Reset</i> where the <i>DFP Source</i> applies <i>vSafe5V</i> on <i>VBUS</i> and the <i>UFP Sink</i> is operating at <i>vSafe5V</i> as defined in [USB 2.0] , [USB 3.2] , [USB Type-C 2.4] or [USBBC 1.2] .
<i>USB Device</i>	Either a <i>Hub</i> or a <i>Peripheral</i> device as defined in [USB 2.0] , [USB 3.2] and [USB4] .
<i>USB Host</i>	The computer system where the <i>USB Host</i> controller is installed as defined in [USB 2.0] , [USB 3.2] and [USB4] .
<i>USB Hub</i>	See <i>Hub</i> .
<i>USB Powered State</i>	Synonymous with the [USB 2.0] and [USB 3.2] definition of the powered state.
<i>USB Safe State</i>	State of the <i>USB Type-C</i> connector when there are pins to be re-purposed (see [USB Type-C 2.4]) so they are not damaged by and do not cause damage to their <i>Port Partner</i> .
<i>USB Type-A</i>	Term used to refer to any A plug or receptacle including USB Micro-A plugs and USB Standard-A plugs and receptacles. USB Micro-AB receptacles are assumed to be a combination of <i>USB Type-A</i> and <i>USB Type-B</i> .
<i>USB Type-B</i>	Terms used to refer to any B-plug or receptacle including USB Micro-B plugs and USB Standard-B plugs and receptacles, including the PD and non-PD versions. USB Micro-AB receptacles are assumed to be a combination of <i>USB Type-A</i> and <i>USB Type-B</i> .
<i>USB Type-C</i>	Term used to refer to the <i>USB Type-C</i> connector plug, or receptacle as defined in [USB Type-C 2.4] .
<i>USB Type-C Multi-Port Charger</i>	A product that exposes multiple <i>USB Type-C Source Ports</i> for the purpose of charging multiple connected <i>USB Devices</i> as defined in [USB Type-C 2.4] .
<i>USB-C[®] Port Control</i>	Module in a <i>PD Capable</i> device which controls <i>Attach/Detach</i> and either detects or sets the <i>R_p</i> value.
<i>USB-IF PD SID</i>	Standard ID allocated to this specification by the USB Implementer's Forum.
<i>USB4[®] Mode</i>	Device is operating in a <i>Mode</i> as defined in [USB4] .
<i>UUT</i>	See <i>Unit Under Test</i> .
<i>Variable Supply</i>	A poorly regulated power supply that is not a <i>Battery</i> . This is exposed by the <i>Variable Supply PDO</i> (see Section 6.4.2, "Request Message").
<i>VBUS</i>	The <i>VBUS</i> wire delivers power from a <i>Source</i> to a <i>Sink</i> .
<i>VCONN</i>	Once the connection between <i>USB Host</i> and device is established, the <i>CC</i> pin (CC1 or CC2) in the receptacle that is not connected via the <i>CC</i> wire through the standard cable is re-purposed to source <i>VCONN</i> to power circuits in a <i>Cable Plug</i> , <i>VCONN Powered Accessory</i> or <i>VCONN Powered USB Device</i> (see [USB Type-C 2.4]).
<i>VCONN Powered Accessory</i>	An accessory that is powered from <i>VCONN</i> to operate in an <i>Alternate Mode</i> (see [USB Type-C 2.4]).
<i>VCONN Powered USB Charge Through Device</i>	A <i>CT-VPD</i> is a <i>VPD</i> with an additional port for connecting a <i>Source</i> (e.g., a <i>Charger</i>) as defined in [USB Type-C 2.4] . When no <i>Charger</i> is connected, a <i>CT-VPD</i> behaves as a <i>VPD</i> . When a <i>Charger</i> is connected, no <i>PD</i> communication to the <i>CT-VPD</i> itself is possible as <i>CC</i> is connected to the <i>Charger</i> port. Hence all <i>PD</i> communication then is with the <i>Charger</i> and the cable with which it is connected.

Table 1.4 Terms and Abbreviations (Continued)

Term	Description
<i>VCONN Powered USB Device</i>	<p>A captive cable <i>USB Device</i> that can be powered by either <i>VCONN</i> or <i>VBUS</i> as defined in [USB Type-C 2.4].</p> <p>A <i>VPD</i> is a captive cable <i>USB Device</i> that can be powered by either <i>VCONN</i> or <i>VBUS</i> and only responds to <i>SOP' Communication</i> as defined in the Tables in Section 6.12, "State behavior". It only responds to <i>Messages</i> sent with a <i>Specification Revision</i> of at least <i>Revision 3.x</i>. A <i>VPD</i> is not allowed to support <i>Alternate Modes</i>.</p> <p>The term <i>VPD</i> refers to either a <i>VPD</i> or a <i>CT-VPD</i> with no <i>Charger</i> connected.</p>
<i>VCONN Source</i>	The <i>USB Type-C Port</i> responsible for sourcing <i>VCONN</i> .
<i>VCONN Swap</i>	Process of exchanging the <i>VCONN Source</i> between <i>Port Partners</i> .
<i>VDEM</i>	See <i>Vendor Defined Extended Message</i> .
<i>VDM</i>	See <i>Vendor Defined Message</i> .
<i>VDM Header</i>	The first <i>Data Object</i> following the <i>Message Header</i> in a <i>Vendor Defined Message</i> . The <i>VDM Header</i> contains the <i>SVID</i> relating to the <i>VDM</i> being sent and provides information relating to the <i>Command</i> in the case of a <i>Structured VDM</i> (see Section 6.4.4, "Vendor Defined Message").
<i>VDO</i>	See <i>Vendor Data Object</i> .
<i>Vendor Data Object</i>	<i>Data Object</i> used to send Vendor specific information as part of a <i>Message</i> .
<i>Vendor Defined Extended Message</i>	<i>PD Extended Message</i> defined for vendor/standards usage. A <i>VDEM</i> does not define any structure and <i>Messages</i> can be created in any manner that the vendor chooses.
<i>Vendor Defined Message</i>	<i>PD Data Message</i> defined for vendor/standards usage. These are further partitioned into <i>Structured Vendor Defined Messages</i> , where <i>Commands</i> are defined in this specification, and <i>Unstructured Vendor Defined Messages</i> which are entirely vendor defined (see Section 6.4.4, "Vendor Defined Message").
<i>Vendor ID</i>	16-bit unsigned value assigned by the USB-IF to a given Vendor.
<i>Version</i>	A minor release of the USB Power Delivery specification associated with a particular <i>Revision</i> . <i>Version</i> numbers are also defined in <i>VDMs</i> .
<i>VI</i>	Same as power (i.e., voltage * current = power)
<i>VID</i>	See <i>Vendor ID</i> .
<i>VPD</i>	See <i>VCONN Powered USB Device</i> .

1.7 Parameter Values

The parameters in this specification are expressed in terms of absolute values. For details of how each parameter is measured in compliance please see [\[USBPDCompliance\]](#).

1.8 Changes from Revision 3.0

Extended Power Range (EPR) including Adjustable Voltage Supply (AVS) has been added.

1.9 Compatibility with Revision 2.0

This Revision of the USB Power Delivery specification is designed to be fully inter-operable with [\[USBPD 2.0\]](#) systems using BMC Signaling over the [\[USB Type-C 2.4\]](#) connector and to be compatible with Revision 2.0 hardware.

Please see [Section 2.3, "USB Power Delivery Capable Devices"](#) for more details of the mechanisms defined to enable compatibility.

2 Overview

This section contains no **Normative** requirements.

2.1 Introduction

USB Power Delivery (PD) defines the mechanisms for pairs of directly *Attached Ports* (also referred to as *Port Partners* or *Port Pairs*) to *Negotiate* voltage, current and/or direction of power flow over the USB cable. It uses the USB Type-C® connector's CC wire as the communications channel. The PD mechanisms operate independently of and supersede other USB methods defined in [USB 2.0], [USB 3.2], [USBBC 1.2] and [USB Type-C 2.4].

USB Power Delivery also defines sideband mechanisms used for configuration management of USB Type-C devices and cables. Using *Structured Vendor Defined Messages* (*Structured VDMs*), PD facilitates discovery of device and cables features and performance. *Structured VDMs* are also used to enter/exit some *Active Modes*, either USB-based (e.g., USB4® Mode) or USB Type-C *Alternate Modes*. *Alternate Modes* are associated with *Standard or Vendor IDs* (*SVIDs*) and can be either standard (e.g., DisplayPort *Alternate Mode*) or proprietary (e.g., Intel Thunderbolt™ 3).

2.1.1 Power Delivery Source Operational Contracts

A PD Source will be in one of three *Contracts*:

- *Default Contract* which it enters immediately following a *Connect* where the *Source* provides 5V and *Advertises* the amount of current it can deliver using the R_p value as defined in [USB Type-C 2.4]. A *Source* in a *Default Contract* will remain in this *Contract* until the *Sink* is *Detached* or the *Source* and *Sink* *Negotiate* and enter an *Explicit Contract*.
- *Implicit Contract* which immediately follows a *Power Role Swap* or *Fast Role Swap* and is transitory. The PD *Source* provides 5V and *Advertises* the amount of current it can deliver using the R_p value as defined in [USB Type-C 2.4]. A *Source* in an *Implicit Contract* will immediately *Negotiate* with the *Sink* and enter an *Explicit Contract*.
- *Explicit Contract* is the state of the *Source* after any PD power *Negotiation* consisting of the *Source* sending a *Source_Capabilities* Message, the *Sink* responding with a *Request* Message, the *Source* acknowledging the request with an *Accept* Message and finally the *Source* sends a *PS_RDY* Message when the *Source* is ready to deliver the requested power. This is the normal operational state for PD. A *Source* in an *Explicit Contract* will remain in an *Explicit Contract* during and after a *Re-negotiation* of its *Contract* and will exit the *Explicit Contract* when:
 - Disconnected from the *Sink* where it will restart in a *Default Contract* when reconnected to the *Sink*.
 - Following a *Hard Reset* where it will restart as if it were *Detached* then *Attached* to the *Sink*.
 - Following a *Power Role Swap* or *Fast Role Swap* where it will enter an *Implicit Contract*.
 - Following USB Type-C Error Recovery which is an electrical *Detach/Re-attach* (remove and assert R_p).

2.1.2 Power Delivery Contract Negotiation

Contracts Negotiated using the USB Power Delivery Specification supersede any and all previous power contracts established whether from standard [USB 2.0], [USB 3.2], [USBBC 1.2] or [USB Type-C 2.4] mechanisms. While operating in *Power Delivery Mode* there will be a *Contract* in place (either *Explicit Contract* or *Implicit Contract*) that determines the power level available and the direction of that power. The *Port Pair* will remain in *Power Delivery Mode* until the *Port Pair* is *Detached*, there is a *Hard Reset*, or USB Type-C Error Recovery, or the *Source* removes power except as part of the *Power Role Swap* or *Fast Role Swap* processes.

Note: [USB4] does not define a default power, rather relies on a PD power *Contract*. When first *Attached* the [USB4] device operates in [USB 3.2] Mode which is its USB Default Operation.

An *Explicit Contract* is *Negotiated* by the process of the *Source* sending a set of *Capabilities*, from which the *Sink* is required to request a particular capability and then the *Source* accepting this request.

An *Implicit Contract* is the specified level of power allowed in particular states (i.e., during and after a *Power Role Swap* or *Fast Role Swap*). *Implicit Contracts* are temporary; *Port Pairs* are required to immediately *Negotiate* an *Explicit Contract*.

Each *Provider* has a *Local Policy*, governing power allocation to its *Ports*. *Consumers* also have their own *Local Policy* governing how they draw power. A *System Policy* can be enacted over USB that allows modification to this *Local Policy* and hence management of overall power allocation in the system.

When *PD Capable* devices are *Attached* to each other, the *DFPs* and *UFPs* initially default to standard *USB Default Operation*. The *DFP* supplies *vSafe5V* and the *UFP* draws current in accordance with the rules defined by [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#) specifications. After *Power Delivery Negotiation* has taken place, power can be supplied at higher, or lower, voltages and higher currents than defined in these specifications. It is also possible to:

- Do a *Power Role Swap* or *Fast Role Swap* to exchange the *Power Roles* such that the *DFP* receives power and the *UFP* supplies power.
- Do a *Data Role Swap* such that the *DFP* becomes the *UFP* and vice-versa.
- Do a *VCONN Swap* to change the *Port* supplying *VCONN* to the cable.
- Enter into *EPR Mode*.
- Enter into *USB4[®] Mode*.
- Enter into *Alternate Modes*.
- Send *Vendor Defined Messages*.

Prior to the *First Explicit Contract* only the *Source Port*, which is also the *VCONN Source*, can communicate with the *Attached* cable assembly. This is important where 5A and *EPR* capability are marked as well as other details of the cable assembly such as the supported speed.

Cable Discovery, determining whether the cable can communicate, can occur on initial *Attachment* of a *Port Pair* before an *Explicit Contract* has been established. It is also possible to carry out *Cable Discovery* after a *Power Role Swap* or *Fast Role Swap* prior to re-establishing an *Explicit Contract*, where the *UFP* is the *Source*, and an *Implicit Contract* is in place. *Cable Discovery* can be carried out after an *Explicit Contract* has been established, if the cable has not yet been discovered.

2.1.3 Other Uses for Power Delivery

Once an *Explicit Contract* is in place, *PD* can be used to manage the *Ports* and cables for non-power related functionality.

PD is used to enter the *USB4[®] Mode* of operation. *Ports* and cables can support functionality beyond power. For example, a cable can have active components that require *VCONN* power or a *Port/cable* can support a video display *Alternate Mode* such as DisplayPort. *PD* defines an infrastructure to discover these additional *Capabilities* and *Modes* that include:

- Discovering a *Port* or *Cable Plug's Capabilities*.
- Discovery of the *SVIDs* a *Port* or *Cable Plug* supports.
- Discovery of the *Modes* a *Port* or *Cable Plug* supports.
- Entry into a *Mode* supported by the *Port* and/or *Cable Plug*.
- Exiting *Modes* supported by the *Port* and/or *Cable Plug*.

2.2 Compatibility with Revision 2.0

Revision 3.x of the USB Power Delivery specification is designed to be fully inter-operable with [\[USBPD 2.0\]](#) systems using *BMC Signaling* over the [\[USB Type-C 2.4\]](#) connector and to be compatible with *Revision 2.0* hardware.

This specification mandates that all *Revision 3.x* systems fully support *Revision 2.0* operation. They must discover the supported *Revision* used by their *Port Partner* and any connected *Cable Plugs* and revert to operation using the lowest common *Revision* number (see [Section 6.2.1.1.5, "Specification Revision"](#)).

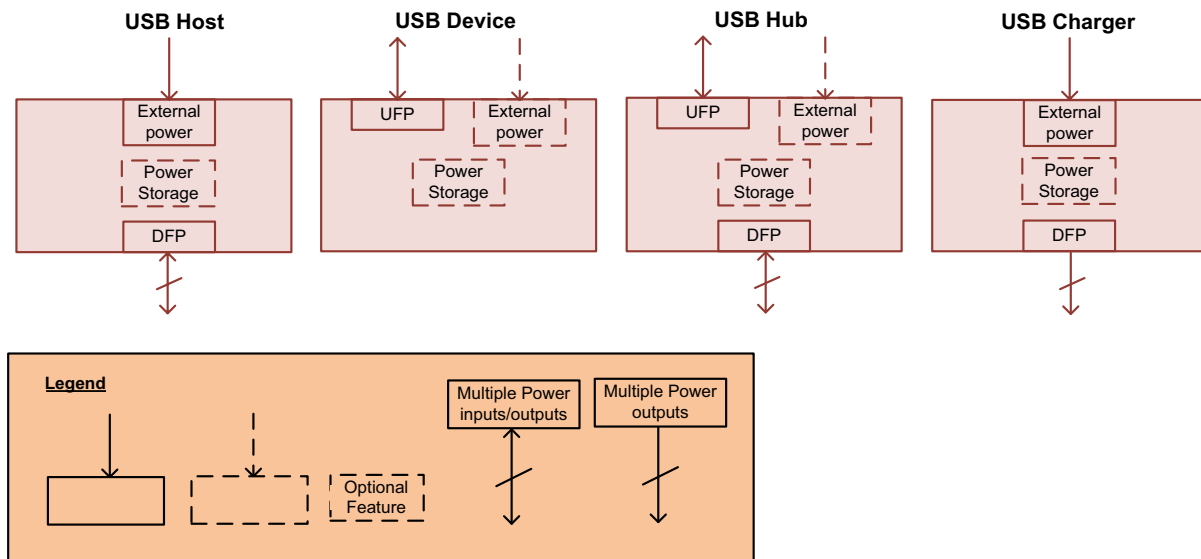
This specification defines *Extended Messages* containing data of up to 260 bytes (see [Section 6.2.1.2, "Extended Message Header"](#)). These *Messages* can be larger than expected by existing PHY HW. To accommodate *Revision 2.0* based systems a *Chunking* mechanism is mandated such that *Messages* are limited to *Revision 2.0* sizes unless it is discovered that both systems support the longer *Message* lengths.

This specification includes changes to the *Vendor Data Objects* (VDO) used in the discovery of passive/active marked cables and *Alternate Mode Adapters* (AMA) (see [Section 6.4.4.2, "Structured VDM"](#)). To enable systems to determine which VDO format is being used the *Structured Vendor Defined Message* (SVDM) *Version* number has been incremented to 2.x. *Version* numbers have also been incorporated into the VDOs themselves to facilitate future changes if these become necessary.

2.3 USB Power Delivery Capable Devices

Some examples of USB Power Delivery capable devices can be seen in [Figure 2.1, "Logical Structure of USB Power Delivery Capable Devices"](#) (a *USB Host*, a *USB Device*, a *Hub*, and a *Charger*). These are given for reference only and are not intended to limit the possible configurations of products that can be built using this specification.

Figure 2.1 Logical Structure of USB Power Delivery Capable Devices



Each USB Power Delivery capable device is assumed to be made up of at least one *Port*. *Providers* are assumed to have a *Source* and *Consumers* a *Sink*. Each device contains one, or more, of the following components:

- *UFPs* that:
 - Sink Power.
 - Communicate using *SOP Packets*.
 - **Optionally** Communicate using *SOP' Packets/SOP'' Packets*.
 - **Optionally** source power (a *Dual-Role Power Device*).
 - **Optionally** communicate via USB.
 - **Optionally** support *Alternate Modes*.
- *DFPs* that:

- Source Power
- Communicate using *SOP Packets*.
- **Optionally** Communicate using *SOP* Packets*.
- **Optionally** Sink power (a *Dual-Role Power Device*).
- **Optionally** communicate via USB.
- **Optionally** support *Alternate Modes*.
- A *Source* that can be:
 - An externally powered source (e.g., AC powered).
 - Power Storage (e.g., *Battery/Power Bank*).
 - Derived from another *Port* (e.g., bus-powered *Hub*).
- A *Sink* that can be:
 - Power Storage (e.g., a *Battery/Power Bank*).
 - Used to power internal functions.
 - Used to power devices *Attached* to other devices (e.g., a bus-powered *Hub*).
- A *VCONN Source* that:
 - Can be either *Port Partner*, either the *DFP/UFP* or *Source/Sink*.
 - Powers the *Cable Plug(s)*.
 - Powers *VPDs* (*VCONN Powered USB Devices*).
 - Is the only *Port* allowed to talk to the *Cable Plug(s)* at any given time.

2.4 SOP* Communication

2.4.1 Introduction

The *Start of Packet* (or *SOP*) is used as an addressing scheme to identify whether the communications were intended for one of the *Port Partners* (*SOP Communication*) or one of the *Cable Plugs* (*SOP' Communication/SOP'' Communication*). *SOP/SOP'* and *SOP''* are collectively referred to as *SOP**. All *SOP* Communications* take place over a single wire (*CC*). The term *Cable Plug* in the *SOP' Communication/SOP'' Communication* case is used to represent a logical entity in the cable which is capable of *PD Communication*, and which might or might not be physically located in the plug.

Note: There are other *SOPs* defined for special operation such as debug which are not discussed here.

The following sections describe how this addressing scheme operates for *Port-to-Port* and *Port to Cable Plug* communication.

2.4.2 SOP* Collision Avoidance

For all *SOP** the *Source* co-ordinates communication to avoid bus collisions by allowing the *Sink* to initiate messaging when it does not need to communicate itself. Once an *Explicit Contract* is in place, the *Source* manipulates its R_p value (3A) to indicate to the *Sink* that it can initiate an *Atomic Message Sequence (AMS)*. This *AMS* can be communication with the *Source* or with one of the *Cable Plugs*. As soon as the *Source* itself needs to initiate an *AMS*, it will manipulate its R_p value (1.5A) to indicate this to the *Sink*. The *Source* then waits for any outstanding *Sink SOP* Communication* to complete before initiating an *AMS* itself. In all cases, the *Port* initiating an *AMS* waits for *CC* to be *Idle* before putting the *Message* on *CC*.

2.4.3 SOP Communication

SOP Communication is used for *Port-to-Port* communication between the *Source* and the *Sink*. *SOP Communication* is recognized by both *Port Partners* but not by any intervening *Cable Plugs*. *SOP Communication* takes priority over other *SOP* Communications* since it is critical to complete power related operations as soon as possible.

2.4.4 SOP'/SOP'' Communication with Cable Plugs

SOP' Communication is recognized by electronics in one *Cable Plug* (see [\[USB Type-C 2.4J\]](#)). *SOP'' Communication* can also be supported when *SOP' Communication* is also supported. *SOP'* and *SOP''* assignment in the cable assembly is fixed and does not change dynamically.

SOP Communication between the *Port Partners* is not recognized by the *Cable Plug*. [Figure 2.2, "Example SOP' Communication between VCONN Source and Cable Plug\(s\)"](#) outlines the usage of *SOP* Communications* between a *VCONN Source (DFP/UFP)* and the *Cable Plugs*.

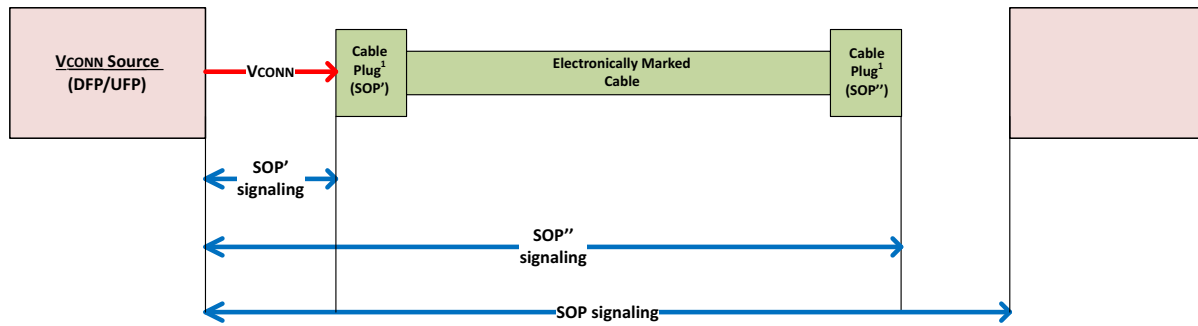
Since all *SOP* Communications* take place over a single wire (*CC*), the *SOP* Communication* periods must be coordinated to prevent important communication from being blocked. For a product which does not recognize *SOP/SOP'* or *SOP'' Packets*, this will look like a non-*Idle* channel, leading to missed *Packets* and retries. Communications between the *Port Partners* take precedence meaning that communications with the *Cable Plug* can be interrupted but will not lead to a *Soft Reset* or *Hard Reset*.

When a *Default Contract* or *Implicit Contract* is in place (e.g., at startup, after a *Power Role Swap* or *Fast Role Swap*) only the *Source Port* that is supplying *VCONN* is allowed to send *Packets* to a *Cable Plug (SOP')* and is allowed to respond to *Packets* from the *Cable Plug (SOP')* with a *GoodCRC Message* in order to discover the *Cable Plug's* characteristics (see [Figure 2.2, "Example SOP' Communication between VCONN Source and Cable Plug\(s\)"](#)). During this phase, all communication with the *Cable Plug* is initiated and controlled by the *VCONN Source* which acts to prevent conflicts between *SOP Packets* and *SOP' Packets*. The *Sink* does not communicate with the *Cable Plug* and **Discards** any *SOP' Packets* received.

When an *Explicit Contract* is in place, only the *VCONN Source* (either the *DFP* or the *UFP*) can communicate with the *Cable Plug(s)* using *SOP' Packets/SOP'' Packets* (see [Figure 2.2, "Example SOP' Communication between VCONN Source and Cable Plug\(s\)"](#)). During this phase, all communication with the *Cable Plug* is initiated and controlled by the *VCONN Source* which acts to prevent conflicts between *SOP* Packets*. The *Port* that is not the *VCONN Source* is not

allowed to communicate with the *Cable Plug* and does not recognize any *SOP' Packets*/*SOP'' Packets* received. Only the *DFP*, when acting as a *VCONN Source*, is allowed to send *SOP* Packets* to control the entry and exiting of *Modes* and to manage *Modal Operation*.

Figure 2.2 Example SOP' Communication between VCONN Source and Cable Plug(s)



2.5 Operational Overview

A USB Power Delivery *Port* supplying power is known as a *Source* and a *Port* consuming power is known as a *Sink*. There is only one *Source Port* and one *Sink Port* in each *PD Connection* between the *Port Partners*. At *Attach* the *Source Port* (the *Port* with R_p asserted see [USB Type-C 2.4]) is also the *DFP* and *VCONN Source*. At *Attach* the *Sink Port* (the *Port* with R_d asserted) is also the *UFP* and is not the *VCONN Source*.

The original USB *PD* specification allowed *Sources* to deliver up to 100W. This classic *Mode* of operation is referred to as the *Standard Power Range (SPR)*. The *First Explicit Contract*, the first *Contract* after a *Default Contract* or *Implicit Contract*, is always an *SPR Contract*. There is an **Optional** higher power *Mode* referred to as the *Extended Power Range (EPR)* where the *Source* is allowed to deliver up to 240W. The *EPR Mode* can only be entered from the *SPR Mode*. The entry process is designed to prevent accidental entry into this higher power *Mode*. It can be entered only when an *SPR Explicit Contract* is in place and both the *Source Port* and *Sink Port* as well as the cable support *EPR*.

The *Source/Sink Power Roles*, *DFP/UFP Data Roles* and *VCONN Source* role can all subsequently be swapped orthogonally to each other. A *Port* that supports both *Source* and *Sink Power Roles* is called a *Dual-Role Power Port (DRP)*. A *Port* that supports both *DFP* and *UFP Data Roles* is called a *Dual-Role Data Port (DRD)*.

When *USB Communications* capability is supported in the *DFP Data Role* then the *Port* will also be able to act as a *USB Host*. Similarly, when *USB Communications* capability is supported in the *UFP Data Role* then the *Port* will also be able to act as a *USB Device*.

The following sections describe the high-level operation of ports taking on the roles of *DFP*, *UFP*, *Source* and *Sink*.

For details of how *PD* maps to USB states in a *PDUSB Device* see [Section 9.1.2, "Mapping to USB Device States"](#).

2.5.1 Source Operation

The *Source* operates differently depending on its *Attachment* status:

- At *Attach* (no *PD Connection* or *Contract*):
 - For a *Source-only Port* the *Source* detects *Sink Attachment*.
 - For a *DRP* that toggles between *Source* and *Sink* operation, the *Port* becomes a *Source Port* on *Attachment* of a *Sink*
 - The *Source* then supplies **vSafe5V**.
- Before *PD Connection* (no *PD Connection* or *Contract*):
 - Prior to sending **Source_Capabilities** Messages the *Source* can detect the *Cable Capabilities* and Advertises its *Capabilities* depending on the *Cable Capabilities* detected:
 - The default current carrying capability of a *USB Type-C* cable is 3A.
 - The *Source* can attempt to communicate with one of the *Cable Plugs* using *SOP' Packets*. If the *Cable Plug* responds, then communication takes place to discover the cable's *Capabilities* (e.g., 5A capable).
 - The *Source* periodically Advertises its *Capabilities* by sending **Source_Capabilities** Messages every **tTypeCSourceCap**.
- Establishing *PD Connection* (no *PD Connection* or *Contract*):
 - Presence of a *PD Capable Port Partner* is detected either:
 - By receiving a **GoodCRC** Message in response to a **Source_Capabilities** Message.
 - By receiving **Hard Reset** Signaling.
- Establishing the *First Explicit Contract* after an *Attach*, *Hard Reset*, *USB Type-C Error Recovery* or *Implicit Contract* as a result of a *Power Role Swap* or *Fast Role Swap*:

- The *Source* receives a **Request** Message from the *Sink* and, if this is a **Valid** request, responds with an **Accept** Message followed by a **PS_RDY** Message when its power supply is ready to source power at the agreed level. At this point an *Explicit Contract* has been agreed.
- A *DFP* that does not generate *SOP' Packets* or *SOP'' Packets*, is not required to detect *SOP' Packets* or *SOP'' Packets* and **Discards** them.
- When in an *Explicit Contract* (**PE_SRC_Ready** State):
 - The *Source* processes and responds (if a response is required) to all *Messages* received and sends appropriate *Messages* whenever its *Local Policy* requires:
 - The *Source* informs the *Sink* whenever its *Capabilities* change, by sending a **Source_Capabilities** Message.
 - The *Source* responds to a *Sink Request* Message with the *Capabilities* mismatch bit set, by sending a **Source_Capabilities** Message with its maximum available power.
 - The *Source* will always have an R_p value asserted on its *CC* wire used for *Collision Avoidance*.
 - When this *Port* is a *DRP* the *Source* can initiate or receive a request for the exchange of *Power Roles*. After the *Power Role Swap* this *Port* will be a *Sink* and in an *Implicit Contract* until an *Explicit Contract* is *Negotiated* immediately afterwards.
 - When this *Port* is a *DRD* the *Source* can initiate or receive a request for an exchange of *Data Roles*. After a *Data Role Swap* the *DFP* (*USB Host*) becomes a *UFP* (*USB Device*). The *Port* remains a *Source* and the *VCONN Source* role remains unchanged.
 - The *Source* can initiate or receive a request for an exchange of *VCONN Source* role. During a *VCONN Swap* *VCONN* is applied by both *Ports* (make before break). The *Port* remains a *Source* and *DFP/UFP Data Roles* remain unchanged.
 - The *Source* when it is the *VCONN Source* can communicate with a *Cable Plug* using *SOP' Communication* or *SOP'' Communication* at any time it is not engaged in any other *SOP Communication*:
 - If *SOP Packets* are received by the *Source*, during *SOP' Communication* or *SOP'' Communication*, the *SOP' Communication* or *SOP'' Communication* is immediately terminated (the *Cable Plug* times out and does not retry).
 - If the *Source* needs to initiate an *SOP Communication* during an ongoing *SOP' Communication* or *SOP'' Communication* (e.g., for a *Capabilities* change) then the *SOP' Communication* or *SOP'' Communications* will be interrupted.
 - When the *Source Port* is also a *DFP*:
 - The *Source* can control the entry and exiting of *Modes* in the *Cable Plug(s)* and control *Modal Operation*.
 - The *Source* can initiate *Unstructured VDMs* or *Structured VDMs*.
 - The *Source* can control the entry and exiting of *Modes* in the *Sink* and control *Modal Operation* using *Structured VDMs*.
- *Detach* or communications failure:
 - A *Source* detects plug *Detach* and takes *VBUS* down to **vSafe5V** within **tSafe5V** and **vSafe0V** within **tSafe0V** (i.e. using *USB Type-C Detach* detection via *CC*).
 - When the *Source* detects the failure to receive a **GoodCRC** Message in response to a *Message* within **tReceive**:
 - Leads to a *Soft Reset*, within **tSoftReset** of the **CRCReceiveTimer** expiring.

- If the *Soft Reset* process cannot be completed a *Hard Reset* will be issued within *tHardReset* of the *CRCReceiveTimer* to restore *VBUS* to *USB Default Operation* within ~1-1.5s:
 - When the *Source* is also the *VCONN Source*, *VCONN* will also be power cycled during the *Hard Reset*.
- When the *Source* operating in *SPR PPS Mode* fails to receive periodic communication (e.g., a *Request Message*) from the *Sink* within *tPPSTimeout*:
 - *Source* issues a *Hard Reset* and takes *VBUS* to *vSafe5V*.
- When the *Source* operating in the *EPR Mode* fails to receive periodic communication (i.e., an *EPR_KeepAlive Message* or any other *Message*) from the *Sink* within *tSourceEPRKeepAlive*:
 - *Source* issues a *Hard Reset* and takes *VBUS* to *vSafe5V*.
- Receiving no response to further attempts at communication is interpreted by the *Source* as an error (see Error handling).
- Errors during power transitions will automatically lead to a *Hard Reset* to restore power to default levels.
- Error handling:
 - *Protocol Errors* are handled by a *Soft_Reset Message* issued by either *Port Partner*, that resets counters, timers and states, but does not change the *Negotiated* voltage and current or the *Port's* role (e.g., *Source*, *DFP/UFP*, *VCONN Source*) and does not cause an exit from *Modal Operation*.
 - Serious errors are handled by *Hard Reset Signaling* issued by either *Port Partner*. A *Hard Reset*:
 - Resets protocol as for a *Soft Reset* but also returns the power supply to *USB Default Operation* (*vSafe0V* or *vSafe5V* output) in order to protect the *Sink*.
 - Restores the *Port's Data Role* to *DFP*.
 - Restores the *Port's* power to its USB default state.
 - When the *Sink* is the *VCONN Source* it removes *VCONN* then the *Source Port* is restored as the *VCONN Source*.
 - Causes all *Active Modes* to be exited such that the *Source* is no longer in *Modal Operation*.
 - After a *Hard Reset* it is expected that the *Port Partner* will respond within *tNoResponse*. If this does not occur then *nHardResetCount* further *Hard Resets* are carried out before the *Source* performs additional *Error Recovery* steps, as defined in [USB Type-C 2.4], by entering the *ErrorRecovery* state.

2.5.2 Sink Operation

- At Attach (no PD Connection or Contract):
 - Sink detects Source Attachment through the presence of **vSafe5V**.
 - For a DRP that toggles between Source and Sink operation, the Port becomes a Sink Port on Attachment of a Source.
 - Once the Sink detects the presence of **vSafe5V** on VBUS it waits for a **Source_Capabilities** Message indicating the presence of a PD Capable Source.
 - If the Sink does not receive a **Source_Capabilities** Message within **tTypeCSinkWaitCap** then it can issue **Hard Reset** Signaling in order to cause the Source Port to send a **Source_Capabilities** Message if the Source Port is PD Capable.
 - The Sink does not generate SOP' Packets or SOP'' Packets, is not required to detect SOP' Packets or SOP'' Packets and **Discards** them.
- Establishing PD Connection (no PD Connection or Contract):
 - The Sink receives a **Source_Capabilities** Message and responds with a **GoodCRC** Message.
 - The Sink does not generate SOP' Packets or SOP'' Packets, is not required to detect SOP' Packets or SOP'' Packets and **Discards** them.
- Establishing the First Explicit Contract after an Attach, Hard Reset or Implicit Contract as a result of a Power Role Swap or Fast Role Swap:
 - The Sink receives a **Source_Capabilities** Message from the Source and responds with a **Request** Message. If this is a **Valid** request the Sink receives an **Accept** Message followed by a **PS_RDY** Message when the Source's power supply is ready to source power at the agreed level. At this point the Source and Sink have entered into an Explicit Contract:
 - The Sink Port can request one of the Capabilities offered by the Source, even if this is the **vSafe5V** output offered by **[USB 2.0]**, **[USB 3.2]**, **[USB Type-C 2.4]** or **[USBBC 1.2]**, in order to enable future power Negotiation:
 - A Sink not requesting any **Valid** capability with a **Request** Message results in an error.
 - A Sink unable to fully operate at the offered Capabilities requests the default capability but indicates that it would prefer another power level by setting the **Capability Mismatch** bit in the **Request** Message and also providing a physical indication of the failure to the end user (e.g., using an LED).
 - The Sink does not generate SOP' Packets or SOP'' Packets, is not required to detect SOP' Packets or SOP'' Packets and **Discards** them.
- During PD Connection (Explicit Contract - **PE_SNK_Ready** state):
 - The Sink processes and responds (if a response is required) to all Messages received and sends appropriate Messages whenever its Local Policy requires.
 - A Sink whose power needs have changed indicates this to the Source with a new **Request** Message. The Sink Port can request one of the Capabilities previously offered by the Source, even if this is the **vSafe5V** output offered by **[USB 2.0]**, **[USB 3.2]**, **[USB Type-C 2.4]** or **[USBBC 1.2]**, in order to enable future power Negotiation:
 - Not requesting any capability with a **Request** Message results in an error.
 - A Sink unable to fully operate at the offered Capabilities requests an offered capability but indicates a **Capabilities Mismatch** i.e., that it would prefer another power level also providing a physical indication of the failure to the end user (e.g., using an LED).

- A Sink operating in the *SPR PPS Mode* periodically sends *Request Message* within *tPPSRequest* even if its request is unchanged.
- A Sink operating in the *EPR Mode* periodically communicates with the Source (i.e., sends an *EPR_KeepAlive Message* or any other Message) within *tSourceEPRKeepAlive*.
- The Sink will always have R_d asserted on its CC wire.
- When this Port is a *DRP*, the Sink can initiate or receive a request for the exchange of *Power Roles*. After the *Power Role Swap* this Port will be a *Source* and an *Implicit Contract* will be in place until an *Explicit Contract* is *Negotiated* immediately afterwards.
- When this Port is a *DRD* the Sink can initiate or receive a request for an exchange of *Data Roles*. After a *Data Role Swap* the *UFP (USB Device)* becomes a *DFP (USB Host)*. The Port remains a Sink and *VCONN Source* role (or not) remains unchanged.
- The Sink can initiate or receive a request for an exchange of *VCONN Source*. During a *VCONN Swap* *VCONN* is applied by both ends (make before break). The Port remains a Sink and *DFP/UFP Data Roles* remain unchanged.
- The Sink when it is the *VCONN Source* can communicate with a *Cable Plug* using *SOP' Communication* or *SOP'' Communication* at any time it is not engaged in any other *SOP Communication*:
 - If *SOP Packets* are received by the Sink, during *SOP' Communication* or *SOP'' Communication*, the *SOP' Communication* or *SOP'' Communication* is immediately terminated (the *Cable Plug* times out and does not retry)
 - If the Sink needs to initiate an *SOP Communication* during an ongoing *SOP' Communication* or *SOP'' Communication* (e.g., for a *Capabilities* change) then the *SOP' Communication* or *SOP'' Communications* will be interrupted.
- When the Sink Port is also a *DFP*:
 - The Sink can initiate *Unstructured VDMs* or *Structured VDMs*.
 - The Sink can control the *Mode Entry* and *Mode Exit* of *Modes* in the Source and control *Modal Operation* (e.g. [USB4]).
- *Detach or Communications Failure*:
 - A Sink detects the removal of *VBUS* and interprets this as the end of the *PD Connection*:
 - This is unless the *vSafe0V* is due to either a *Hard Reset*, *Power Role Swap* or *Fast Role Swap*.
 - A Sink detects plug removal (i.e., absence of R_p or *VBUS*) and discharges *VBUS*.
 - When the Sink detects the failure to receive a *GoodCRC Message* in response to a Message within *tReceive*:
 - Leads to a *Soft Reset*, within *tSoftReset* of the *CRCReceiveTimer* expiring.
 - If the *Soft Reset* process cannot be completed a *Hard Reset* will be issued within *tHardReset* of the *CRCReceiveTimer* to restore *VBUS* to *USB Default Operation* within ~1-1.5s.
 - Receiving no response to further attempts at communication is interpreted by the Sink as an error (see Error handling).
 - When the Sink operating in the *SPR PPS Mode* fails to send periodic communication (i.e. a *Request Message*) to the Source within *tPPSRequest*, the Source will issue a *Hard Reset* that results in *VBUS* going to *vSafe5V*.
 - When the Sink operating in the *EPR Mode* fails to send periodic communication (i.e. an *EPR_KeepAlive Message* or any other Message) to the Source within *tSourceEPRKeepAlive* the Source will issue a *Hard Reset* that results in *VBUS* going to *vSafe5V*.

- Errors during power transitions will automatically lead to a *Hard Reset* to restore power to default levels.
- Error handling:
 - *Protocol Errors* are handled by a **Soft_Reset** Message issued by either *Port Partner*, that resets counters, timers and states, but does not change the *Negotiate* voltage and current or the *Port's* role (e.g., *Sink*, *DFP/UFP*, *VCONN Source*) and does not cause an exit from *Modal Operation*.
 - Serious errors are handled by **Hard_Reset** Signaling issued by either *Port Partner*. A *Hard Reset*:
 - resets protocol as for a *Soft Reset* but also returns the power supply to *USB Default Operation* (**vSafe0V** or **vSafe5V** output) in order to protect the *Sink*.
 - restores the *Port's Data Role* to *UFP*.
 - when the *Sink* is the *VCONN Source* it removes *VCONN* then the *Source Port* is restored as the *VCONN Source*.
 - causes all *Active Modes* to be exited such that the *Source* is no longer in *Modal Operation*.
- After a *Hard Reset* it is expected that the *Port Partner* will respond within **tTypeCSinkWaitCap**. If this does not occur, then two further *Hard Resets* are carried out before the *UFP* stays in the **PE_SNK_Wait_for_Capabilities** state.

2.5.3 Cable Plugs

- *Cable Plugs* are powered when *VCONN* is present but are not aware of the status of the *Contract* between the ports the cable assembly is connecting.
- *Cable Plugs* do not initiate *AMs* and only respond to *Messages* sent to them.
- *Detach* or Communications Failure:
 - Communications can be interrupted at any time.
 - There is no communication timeout scheme between the *DFP/UFP* and *Cable Plug*.
 - The *Cable Plug* is ready to respond to potentially repeated requests.
- Error handling:
 - The *Cable Plug* detects *Hard Reset Signaling* to determine that the *Source* and *Sink* have been reset and will need to reset itself (equivalent to a power cycle).
 - The *Cable Plug* cannot generate *Hard Reset Signaling* itself.
 - The *Hard Reset* process power cycles both *VBUS* and *VCONN* so this is expected to reset the *Cable Plugs* by itself.
- A *Cable Plug* detects *Cable Reset Signaling* to determine that it will need to reset itself (equivalent to a power cycle).

2.6 Architectural Overview

This logical architecture is not intended to be taken as an implementation architecture. An implementation architecture is, by definition, a part of product definition and is therefore outside of the scope of this specification.

This section outlines the high-level logical architecture of USB Power Delivery referenced throughout this specification. In practice various implementation options are possible based on many different possible types of *PD* devices. *PD* devices can have many different configurations e.g., *USB Communication* or non-*USB Communication*, single versus multiple *Ports*, dedicated power supplies versus supplies shared on multiple ports, hardware versus software-based implementations etc. The architecture outlined in this section is therefore provided only for reference to indicate the high-level logical model used by the *PD* specification. This architecture is used to identify the key concepts and to indicate logical blocks and possible links between them.

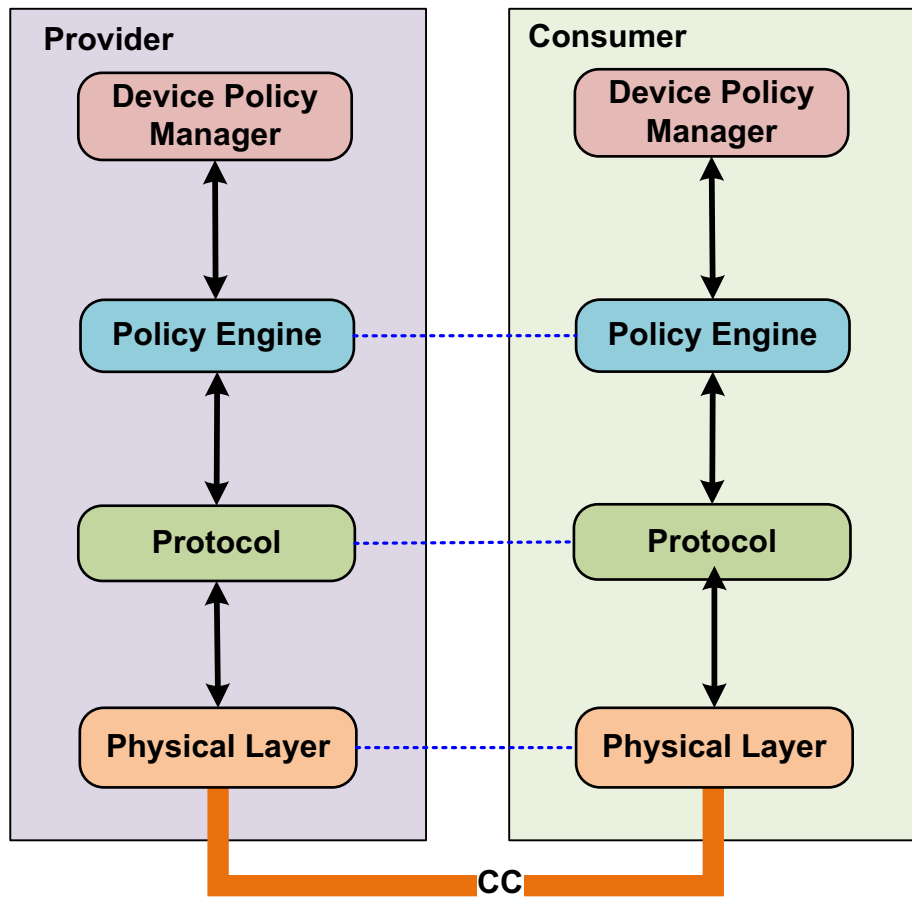
The USB Power Delivery is a *Port* to *Port* architecture in which each *PD Capable* device is made up of several major components.

- [Figure 2.3, "USB Power Delivery Communications Stack"](#) illustrates the relationship of the layers of the communications stack between a *Port Pair*.

The communications stack consists of:

- A *Device Policy Manager* (see [Section 8.2, "Device Policy Manager"](#)) that exists in all devices and manages USB Power Delivery resources within the device across one or more *Ports* based on the device's *Local Policy*.
- A *Policy Engine* (see [Section 8.3, "Policy Engine"](#)) that exists in each USB Power Delivery *Port* implements the *Local Policy* for that *Port*.
- A *Protocol Layer* (see [Section 6, "Protocol Layer"](#)) that enables *Messages* to be exchanged between a *Source Port* and a *Sink Port*.
- A *PHY Layer* (see [Section 5, "Physical Layer"](#)) that handles transmission and reception of bits on the wire and handles data transmission

Figure 2.3 USB Power Delivery Communications Stack



Additionally, USB Power Delivery devices which can operate as USB devices can communicate over USB (see [Figure 2.4, "USB Power Delivery Communication Over USB"](#)). An **Optional** System Policy Manager (see [Chapter 9](#) and [\[UCSI\]](#)) that resides in the *USB Host* communicates with the *PDUSB Device* over USB, via the root *Port* and potentially manages the individual *Port* to *Port* connections over a tree of *USB Hubs*. The *Device Policy Manager* interacts with the USB interface in each device to provide and update *PD* related information in the USB domain.

Note: A *PD* device is not required to have a USB device interface.

Figure 2.4 USB Power Delivery Communication Over USB

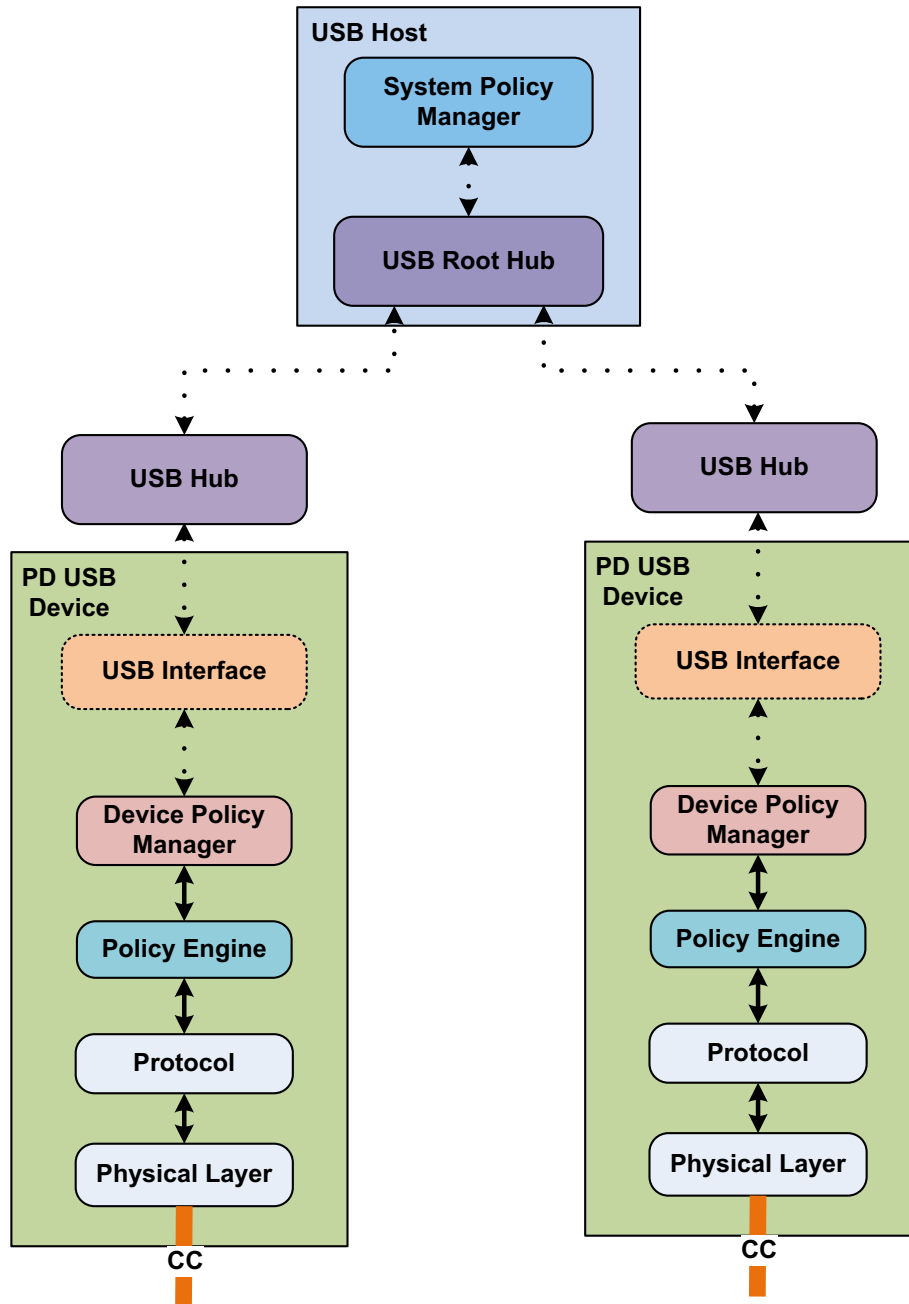


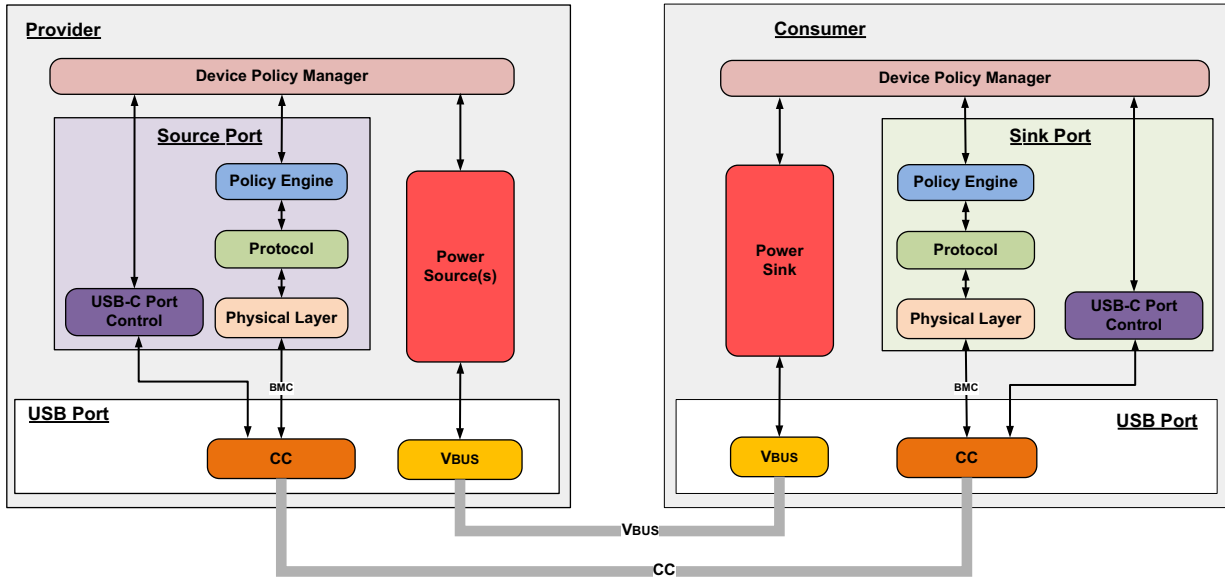
Figure 2.5, "High Level Architecture View" shows the logical blocks between two Attached PD Ports (Port Pair). In addition to the communication stack described above there are also:

- For a *Provider* or *Dual-Role Power Device*: one or more *Sources* providing power to one or more *Ports*.
- For a *Consumer* or *Dual-Role Power Device*: A *Sink* consuming power.
- A USB-C® Port Control module (see [Section 4.4 "Cable Type Detection"](#)) that detects cable *Attach/Detach* as defined in [\[USB Type-C 2.4\]](#).
- USB Power Delivery uses standard cabling as defined in [\[USB Type-C 2.4\]](#).

The *Device Policy Manager* talks to the communication stack, *Source/Sink*, and the *USB-C® Port Control* block to manage the resources in the *Provider* or *Consumer*.

[Figure 2.5, "High Level Architecture View"](#) illustrates a *Provider* and a *Consumer*. *Dual-Role Power Devices* can be constructed by combining the elements of both *Provider* and *Consumer* into a single device. *Providers* can also contain multiple *Source Ports* each with their own communications stack and *USB-C® Port Control*.

Figure 2.5 High Level Architecture View



2.6.1 Policy

There are two levels of Policy:

- 1) *System Policy* applied system wide by the *System Policy Manager* across multiple *Providers* or *Consumers*.
- 3) *Local Policy* enforced on a *Provider* or *Consumer* by the *Device Policy Manager* for a device.

Policy comprises several logical blocks:

- *System Policy Manager* (system wide).
- *Device Policy Manager* (one per *Provider* or *Consumer*).
- *Policy Engine* (one per *Source Port* or *Sink Port*).

2.6.1.1 System Policy Manager

Since the USB Power Delivery protocol is *Port to Port*, implementation of a *System Policy* requires communication by an additional data communication mechanism i.e., USB. [\[UCSI\]](#) has been created to define an interface for the *System Policy Manager* to communicate with the *Device Policy Manager*. When present, the *System Policy Manager* monitors and controls *System Policy* between various *Providers* and *Consumers* connected via USB. The *System Policy Manager* resides in the *USB Host* and communicates via USB with the *Device Policy Manager* in each connected *Device*. *Devices* without *USB Communication* capability or are not data connected, will not be able to participate in *System Policy*.

The *System Policy Manager* is **Optional** so USB Power Delivery *Providers* and *Consumers* will operate without it being present. This includes systems where the *USB Host* does not provide a *System Policy Manager* and can also include "headless" systems without any *USB Host*. In those cases where a *USB Host* is not present, USB Power Delivery is useful for charging purposes, or the powering of devices since useful USB functionality is not possible. Where there is a *USB Host*, but no *System Policy Manager*, *Providers* and *Consumers* can *Negotiate* power between

themselves, independently of USB power rules, but are more limited in terms of the options available for managing power.

2.6.1.2 Device Policy Manager

The *Device Policy Manager* provides mechanisms to monitor and control the USB Power Delivery system within a particular *Consumer* or *Provider*. The *Device Policy Manager* enables *Local Policy* to be enforced across the system by communication with the *System Policy Manager*. *Local Policy* is enacted on a per *Port* basis by the *Device Policy Manager's* control of the *Source Ports/Sink Ports* and by communication with the *Policy Engine* and *USB-C® Port Control* for that *Port*. The *Device Policy Manager* is responsible for the sharing algorithm used in *Shared Capacity Chargers* (see [USB Type-C 2.4])

2.6.1.3 Policy Engine

Providers and *Consumers* are free to implement their own *Local Policy* on their directly connected *Source Ports* or *Sink Ports*. These will be supported by *Negotiation* and status mechanisms implemented by the *Policy Engine* for that *Port*. The *Policy Engine* interacts directly with the *Device Policy Manager* to determine the present *Local Policy* to be enforced. The *Device Policy Manager* will also inform the *Policy Engine* whenever there is a change in *Local Policy* (e.g., *Capabilities* change).

2.6.2 Message Formation and Transmission

2.6.2.1 Protocol Layer

The *Protocol Layer* forms the *Messages* used to communicate information between a *Port Pair*. It is responsible for forming *Capabilities Messages*, requests and acknowledgments. Additionally, it forms *Messages* used to swap roles and maintain presence. It receives inputs from the *Policy Engine* indicating which *Messages* to send and indicates the responses back to the *Policy Engine*.

The basic protocol uses a push model where the *Provider* pushes its *Capabilities* to the *Consumer* that in turn responds with a request based on the offering. However, the *Consumer* can asynchronously request the *Provider's* present *Capabilities* and can select another voltage/current.

Extended Messages of up to a *Data Size* of *MaxExtendedMsgLen* can be sent and received provided the *Protocol Layer* determines that both *Port Partners* support this capability. When one of both *Port Partners* do not support *Extended Messages* of *Data Size* greater than *MaxExtendedMsgLegacyLen* then the *Protocol Layer* supports a *Chunking* mechanism to break larger *Messages* into smaller *Chunks* of size *MaxExtendedMsgChunkLen*. All *Ports* that support *Extended Messages* longer than *MaxExtendedMsgLegacyLen* are required to support *Chunking*.

2.6.2.2 PHY Layer

The *PHY Layer* is responsible for sending and receiving *Messages* across the *USB Type-C CC* wire and for managing data. *PD* is a *Multi-Drop* system, sharing *CC* between the *Port Partners* and the *Cable Plug(s)* that implements *Collision Avoidance* and recovery mechanisms. The *PHY Layer* detects errors in the *Messages* using a *CRC*.

2.6.3 Collision Avoidance

2.6.3.1 Policy Engine

The *Policy Engine* in a *Source* will indicate to the *Protocol Layer* the start and end of each *Atomic Message Sequence (AMS)* that the *Source* initiates. The *Policy Engine* in a *Sink* will indicate to the *Protocol Layer* the start of each *AMS* the *Sink* initiates. This enables co-ordination of *AMS* initiation between the *Port Partners*.

2.6.3.2 Protocol Layer

The *Protocol Layer* in the *Source* will request the *PHY* to set the R_p value to *SinkTxOK* when it is not actively sending *Messages*. This indicates to the *Sink* that it can initiate an *AMS* by sending the first *Message* in the sequence. The *Protocol Layer* in the *Source* will request the *PHY Layer* to set the R_p value to *SinkTxNG* to indicate that the *Sink* cannot initiate an *AMS* since the *Source* is about to initiate an *AMS*.

The *Protocol Layer* in the *Sink*, when the *Policy Engine* indicates that an *AMS* is being initiated, will wait for the R_p value to be set to **SinkTxOK** before initiating the *AMS* by sending the first *Message* in the sequence.

2.6.3.3 PHY Layer

The *PHY Layer* in the *Source* will set the R_p value to either **SinkTxOK** or **SinkTxNG** as directed by the *Protocol Layer*. The *PHY Layer* in the *Sink* will detect the present R_p value and inform the *Protocol Layer*.

2.6.4 Power supply

2.6.4.1 Source

Each *Provider* will contain one or more power sources that are shared between one or more *Ports*. These power sources are controlled by the *Local Policy*. *Source Ports* start up in *USB Type-C Operation* where the *Port* applies **vSafe0V** on *VBUS* and returns to this state on *Detach* or after a *Hard Reset*. When the *Source* detects *Attach* events it transitions its output to **vSafe5V**.

2.6.4.2 Sink

Consumers are assumed to have one *Sink* connected to a *Port*. This *Sink* is controlled by *Local Policy*. *Sinks* start up in *USB Default Operation* where the *Port* can operate at **vSafe5V** with USB default specified current levels and return to this state on *Detach* or after a *Hard Reset*.

2.6.4.3 Dual-Role Power Ports

Dual-Role Power Ports have the ability to operate as either a *Source* or a *Sink* and to swap between the two *Power Roles* using *Power Role Swap* or *Fast Role Swap*.

2.6.4.4 Dead Battery or Lost Power Detection

[USB Type-C 2.4] defines mechanisms intended to communicate with and to charge a *Sink* or *DRP* with a *Dead Battery*.

2.6.4.5 VCONN Source

The *Source Port* at *Attach*, is also the *VCONN Source*. The responsibility for sourcing *VCONN* can be swapped between the *Source Ports* and *Sink Ports* in a make before break fashion to ensure that the *Cable Plugs* are continuously powered. To ensure reliable communication with the *Cable Plugs* only the *Port* that is the *VCONN Source* is permitted to communicate with the *Cable Plugs*.

Note: Prior to a *Power Role Swap*, *Data Role Swap* or *Fast Role Swap* each new *Source Port* needs to ensure that it is the *VCONN Source* if it needs to communicate with the *Cable Plugs* after the swap.

2.6.5 DFP/UFP

2.6.5.1 Downstream Facing Port (DFP)

The *Downstream Facing Port* or *DFP* is equivalent in the USB topology to the *Port* a *USB Device* is *Attached* to. The *DFP* will also correspond to the *USB Host* but only if *USB Communication* is supported while acting as a *DFP*. Products such as *Chargers* can be a *DFP* while not having *USB Communication* capability. Only the *DFP* is allowed to control *Alternate Mode* operation.

2.6.5.2 Upstream Facing Port (UFP)

The *Upstream Facing Port* or *UFP* is equivalent in the USB topology to the *Port* on a *USB Device* that is connected to the *USB Host* or *USB Hub's DFP*. The *UFP* will also correspond to the *USB Device* but only if *USB Communication* is supported while acting as a *UFP*. Products which charge can be a *UFP* while not having *USB Communication* capability.

2.6.5.3 Dual-Role Data Ports

Dual-Role Data Ports have the ability to operate as either a *DFP* or a *UFP* and to swap between the two *Data Roles* using *Data Role Swap*.

Note: Products can be *Dual-Role Data Ports* without being *Dual-Role Power Ports* that is they can switch logically between *DFP* and *UFP Data Roles* even if they are *Source-only* or *Sink-only* Ports.

2.6.6 Cable and Connectors

The USB Power Delivery specification assumes certified USB cables and associated detection mechanisms as defined in the [\[USB Type-C 2.4\]](#) specification.

2.6.6.1 USB-C Port Control

The *USB-C[®] Port Control* block provides mechanisms to:

- Inform the *Device Policy Manager* of cable *Attach/Detach* events.
- Inform *Sink's Device Policy Manager* of the R_p value.
- Allow *Source's Device Policy Manager* to set the R_p value.

2.6.7 Interactions between Non-PD, BC, and PD devices

USB Power Delivery only operates when two USB Power Delivery devices are directly connected. When a device finds itself a mixed environment, where the other device does not support the USB Power Delivery Specification, the existing rules on supplying *vSafe5V* as defined in the [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USBBC 1.2\]](#) or [\[USB Type-C 2.4\]](#) specifications are applied.

There are two primary cases to consider:

- The *USB Host (DFP/Source)* is non-*PD* and as such will not send any *Advertisements*. An *Attached PD Capable* device will not see any *Advertisements* and operates using the rules defined in the [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USBBC 1.2\]](#) or [\[USB Type-C 2.4\]](#) specifications.
- The *Device (UFP/Sink)* is non-*PD* and as such will not see any *Advertisements* and therefore will not respond. The *USB Host (DFP/Source)* will continue to supply *vSafe5V* to *VBUS* as specified in the [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USBBC 1.2\]](#) or [\[USB Type-C 2.4\]](#) specifications.

2.6.8 Power Rules

Power Rules define voltages and current ranges that are offered by compliant USB Power Delivery *Sources* and used by a USB Power Delivery *Sink* for a given value of *PDP Rating*. See [Chapter 10 "Power Rules"](#) for further details.

2.7 Extended Power Range (EPR) Operation

Extended Power Range is a *Mode* that provides for up to 240W which is considerably more power than the 100W the original PD specification (*SPR Mode*) offered. It is a *Mode* of operation that can be entered only when an *Explicit Contract* is in place and both the *Ports* and the *Cable Plug(s)* support *EPR*.

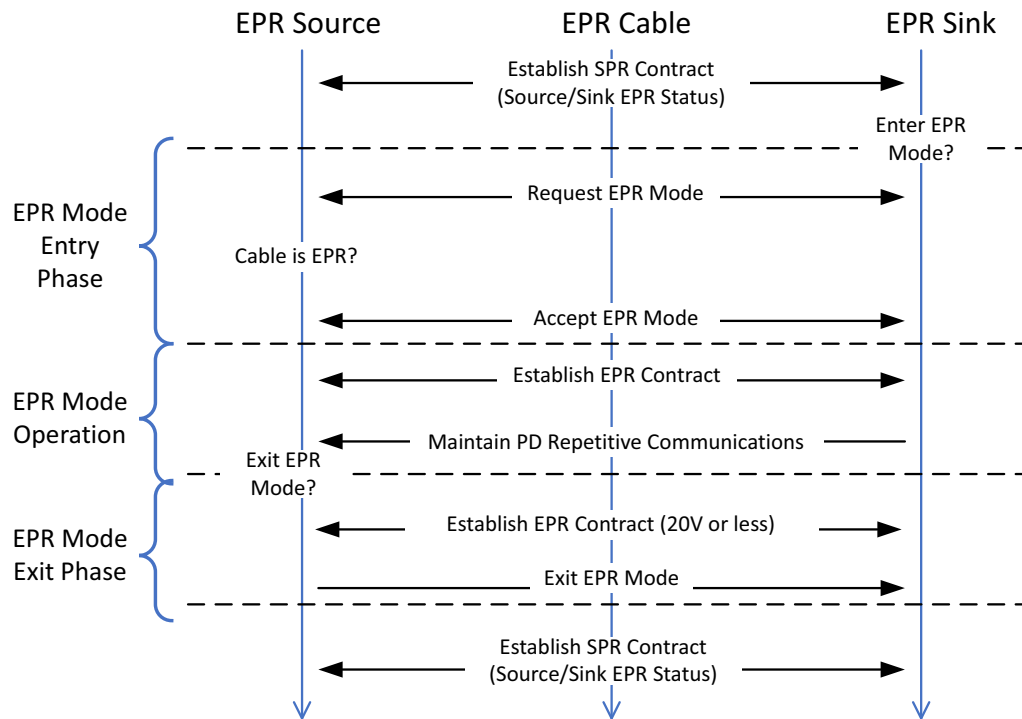
Entry into *EPR Mode* follows a strict process; this assures that the higher voltages, at power levels above 100W, are only transferred between known *EPR Capable Sources* and *EPR Capable Sinks* over *EPR Capable* cables. *EPR Sources* are capable of both *Fixed Supply* and *Adjustable Voltage Supply (AVS)* operation. Maintaining *EPR Mode* operation also requires maintaining a regular cadence of USB PD communications; loss of communications between the *EPR Source* and *EPR Sink* will cause a *Hard Reset* to be initiated resulting in a return to *SPR* operation.

The *EPR Mode* entry, operational and exit process is summarized by the following steps:

- 1) *Negotiate* and enter into an *Explicit Contract* in the *Standard Power Range*. During this step, *EPR Capable Sources* and *Sinks* will declare their supported *EPR Capabilities* through *PDO/APDO* and *RDO* exchanges.
- 2) An *EPR Sink*, having discovered an *EPR Source*, can request *EPR Mode* entry.
- 3) The *EPR Source*, having already confirmed that the *Attached* cable assembly is *EPR Capable* during the *First Explicit Contract Negotiation*, will respond to the *EPR Sink* with an acknowledgment of the *EPR Mode* entry request.
- 4) While in *EPR Mode*:
 - a) The *EPR Source* sends *EPR Capabilities (Fixed Supply PDOs and an AVS APDO)* to the *EPR Sink* which requires the *Sink* to evaluate and respond as appropriate to adjust the *Explicit Contract*.
 - b) The *EPR Sink* maintains a regular cadence of communications with the *EPR Source* to allow *EPR Mode* to continue.
- 5) When either the *EPR Source* or *EPR Sink* no longer wants to remain in *EPR Mode* operation, a normal exit from *EPR Mode* will first require adjusting the *Explicit Contract* to a voltage of 20V or lower (*SPR (A)PDO*) followed by an explicit *EPR Mode* exit request.
 - a) *Source* initiated: *EPR Source* sends an *EPR_Source_Capabilities* Message that only includes *SPR* voltages to force the *EPR Sink* to drop to 20V or below followed by the *EPR Mode* exit. Once *EPR Mode* is exited, a new *SPR Contract* is *Negotiated* to return to *SPR Mode* operation.
 - b) *Sink* initiated; *EPR Sink* requests a drop to 20V or below followed by the *EPR Mode* exit. Once *EPR Mode* is exited, a new *SPR Contract* is *Negotiated* to return to *SPR Mode* operation.

[*Figure 2.6, "Example of a Normal EPR Mode Operational Flow"*](#) illustrates an example of a normal *EPR Mode* operational flow. In this example, at some time during the *EPR Mode* operation, the *Source* decides that it needs to exit *EPR Mode*, so it resends the *EPR Capabilities* to the *Sink* with only *SPR (A)PDOs* to cause the *Sink* to *Negotiate* an *SPR Contract* of 20V or lower and then the *Source* follows with an *EPR Mode exit* Message. Once *EPR Mode* is exited, a new *SPR Contract* is *Negotiated* to return to *SPR Mode* operation.

Figure 2.6 Example of a Normal EPR Mode Operational Flow



Not illustrated in [Figure 2.6, "Example of a Normal EPR Mode Operational Flow"](#), while in *EPR Mode* operation, the *Sink* might decide it wants to exit *EPR Mode*. In this case, the *Sink* must initiate the exit process by revising its *Explicit Contract* with the *Source* at 20V or less followed with an *EPR Mode exit Message*. Once *EPR Mode* is exited, a new *SPR Contract* is *Negotiated* to formalize the return to *SPR Mode* operation. Failure to revise the *Explicit Contract* to one at 20V or less before attempting to exit *EPR Mode* will result in a *Hard Reset*.

2.8 Charging Models

This section provides a charging model overview for each of the primary power delivery methods: *Fixed Supply*, *Programmable Power Supply* and *Adjustable Voltage Supply*.

2.8.1 Fixed Supply Charging Models

USB Power Delivery supports *Fixed Supply* charging using a set of defined standard voltages with current available up to the limit of the *Source's* and cable's *Advertised Capabilities*. As summarized in [Table 2.1, "Fixed Supply Power Ranges"](#), the standard voltages are available in either the *Standard Power Range (SPR)* and/or the *Extended Power Range (EPR)*.

Table 2.1 Fixed Supply Power Ranges

Power Range	Available Current and Voltages	PDP Range	Notes
<i>Standard Power Range (SPR)</i>	3A: 5V, 9V, 15V, 20V 5A ¹ : 20V	15 – 60W >60 – 100W	
<i>Extended Power Range (EPR)</i>	3A ² : 5V, 9V, 15V, 20V 5A ² : 20V 5A ² : 28V, 36V, 48V	15 – 60W >60 – 100W >100 – 240W	Requires entry into <i>EPR Mode</i> .
1) Requires 5A cable.			
2) Requires <i>EPR</i> cable.			

2.8.2 Programmable Power Supply (PPS) Charging Models

USB Power Delivery includes support for *Programmable Power Supply (PPS)* charging using a set of defined standard voltage ranges. With current up to the limit of the *Source's* and cable's *Advertised Capabilities*. Additionally, when operating in *SPR Mode* the current is also limited by the **Operating Current** field value in the **Request Message**.

Note: PPS operation is not available in *EPR Mode*.

The standard voltage ranges available in the *Standard Power Range (SPR)* for PPS are summarized in [Table 2.2, "PPS Voltage Power Ranges"](#).

Table 2.2 PPS Voltage Power Ranges

Available Current	Prog	Min Voltage (V)	Max Voltage (V)	PDP Range
3A	9V Prog	5	11	16 – 60W
	15V Prog	5	16	
	20V Prog	5	21	
5A ¹	20V Prog	5	21	61 – 100W
1) Requires 5A cable.				

2.8.3 Adjustable Voltage Supply (AVS) Charging Models

USB Power Delivery operating in *SPR Mode* (when *PDP* is higher than 27W) and *EPR Mode* includes support for *Adjustable Voltage Supply (AVS)* charging using a set of defined standard voltage ranges based on the *Source's PDP Rating*.

The standard voltage ranges available for *AVS* are summarized in [Table 2.3, "AVS Voltage Power Ranges"](#).

Table 2.3 AVS Voltage Power Ranges

PDP	Minimum Voltage (V)	Maximum Voltage (V)	Maximum Available Current ³	Minimum Voltage (V)	Maximum Voltage (V)	Maximum Available Current
>27...45W	9	15	3A	N/A		
>45...60W	9	20	3A			
>60...100W	9	20	5A ¹			
100...140W	9	20	5A ²	15	28	5A ²
>140...180W	9	20	5A ²	15	36	5A ²
>180...240W	9	20	5A ²	15	48	5A ²
1) Requires 5A cable. 2) Requires an <i>EPR</i> Cable. 3) The maximum available <i>SPR AVS</i> current is determined by the maximum available current in the <i>Fixed Supply</i> 15V <i>PDO</i> in the 9 - 15V range and <i>Fixed Supply</i> 20V <i>PDO</i> in the 15 - 20V range.						

3 USB Type-A and USB Type-B Cable Assemblies and Connectors

This section has been **Deprecated**. Please refer to [\[USBD 2.0\]](#) for details of cables and connectors used in scenarios utilizing the *BFSK Signaling Scheme* in conjunction with *USB Type-A* or *USB Type-B* connectors.

4 Electrical Requirements

This chapter covers the platform's electrical requirements for implementing USB Power Delivery.

4.1 Interoperability with other USB Specifications

USB Power Delivery **May** be implemented alongside the [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB4\]](#), [\[USBBC 1.2\]](#) and [\[USB Type-C 2.4\]](#) (USB Type-C) specifications. In the case where a *Device* requests power via [\[USBBC 1.2\]](#) and then the USB Power Delivery Specification, it **Shall** follow the USB Power Delivery Specification until the *Port Pair* is *Detached* or there is a *Hard Reset*. If the USB Power Delivery connection is lost, the *Port* **Shall** return to its default state, see [Section 6.8.3, "Hard Reset"](#).

4.2 Dead Battery Detection / Unpowered Port Detection

Dead Battery/unpowered operation is when a *USB Device* needs to provide power to a *USB Host* under the circumstances where the *USB Host*:

- Has a *Dead Battery* that requires charging or
- Has lost its power source or
- Does not have a power source or
- Does not want to provide power.

Dead Battery charging operation for connections between *USB Type-C* connectors is defined in [\[USB Type-C 2.4\]](#).

4.3 Cable IR Ground Drop (IR Drop)

Every *PD Sink Port* capable of *USB Communication* can be susceptible to unreliable *USB Communication* if the voltage drop across ground falls outside of the acceptable common mode range for the USB Hi-Speed transceivers data lines due to excessive current draw. Certified USB cabling is specified such that such errors don't typically occur (See [\[USB Type-C 2.4\]](#)).

4.4 Cable Type Detection

Standard *USB Type-C*[®] cable assemblies are rated for *PD* voltages higher than **vSafe5V** and current levels of at least 3A (See [\[USB Type-C 2.4\]](#)). The *Source* **Shall** limit maximum *Capabilities* it offers so as not to exceed the *Capabilities* of the type of cabling detected.

Sources capable of offering more than 3A **Shall** detect the type of *Attached* cable and limit the *Capabilities* they offer based on the current carrying capability of the cable determined by the *Cable Capabilities* determined using the **Discover Identity** Command (see [Section 6.4.4.3.1, "Discover Identity"](#)) sent using *SOP' Communication* (see [Section 2.4, "SOP' Communication"](#)) to the *Cable Plug*. The *Cable VDO* returned as part of the **Discover Identity** Command details the maximum current and voltage values that **Shall** be *Negotiated* for a given cable as part of an *Explicit Contract*.

The *Cable Discovery* process is usually run when the *Source* is powered up, after a *Power Role Swap* or *Fast Role Swap* or when power is applied to a *Sink*. The method used to detect these events **Shall** meet the following requirements:

- *Sources* **Shall** run the *Cable Discovery* process prior to the *Source* sending **Source_Capabilities** Messages offering currents in excess of 3A and/or voltages in excess of 20V.
- *Sinks* with *USB Type-C* connectors **Shall** select *Capabilities* from the offered *Source Capabilities* assuming that the *Source* has already determined the *Capabilities* of the cable.
- *Sinks* with the **Dual-Role Power** bit set, **Shall** respond to a **Get_Source_Cap** Message by declaring their full *Source Capabilities*, without limiting them based on the cable's *Capabilities*.

5 Physical Layer

5.1 Physical Layer Overview

The Physical Layer (*PHY Layer*) defines the *Signaling* technology for USB Power Delivery. This chapter defines the electrical requirements and parameters of the *PHY Layer* required for interoperability between *PDUSB Devices*.

5.2 Physical Layer Functions

The USB PD *PHY Layer* consists of a pair of transmitters and receivers that communicate across a single signal wire (*CC*). All communication is half duplex. The *PHY Layer* practices *Collision Avoidance* to minimize communication errors on the channel.

The transmitter performs the following functions:

- Receive *Packet* data from the *Protocol Layer*.
- Calculate and append a *CRC*.
- Encode the *Packet* data including the *CRC* (i.e., the *Payload*).
- Transmit the *Packet* (*Preamble*, *SOP**, *Payload*, *CRC* and *EOP*) across the channel using *Bi-phase Mark Coding* (*BMC*) over *CC*.

The receiver performs the following functions:

- Recover the clock and lock onto the *Packet* from the *Preamble*.
- Detect the *SOP**.
- Decode the received data including the *CRC*.
- Detect the *EOP* and validate the *CRC*:
 - If the *CRC* is **Valid**, deliver the *Packet* data to the *Protocol Layer*.
 - If the *CRC* is **Invalid**, flush the received data.

5.3 Symbol Encoding

Except for the *Preamble*, all communications on the line **shall** be encoded with a line code to ensure a reasonable level of DC-balance and a suitable number of transitions. This encoding makes receiver design less complicated and allows for more variations in the receiver design.

4b5b line code **shall** be used. This encodes 4-bit data to 5-bit symbols for transmission and decodes 5-bit symbols to 4-bit data for consumption by the receiver.

The 4b5b code provides data encoding along with special symbols. Special symbols are used to signal *Hard Reset*, and delineate *Packet* boundaries (see [Table 5.1, "4b5b Symbol Encoding"](#)).

Table 5.1 4b5b Symbol Encoding

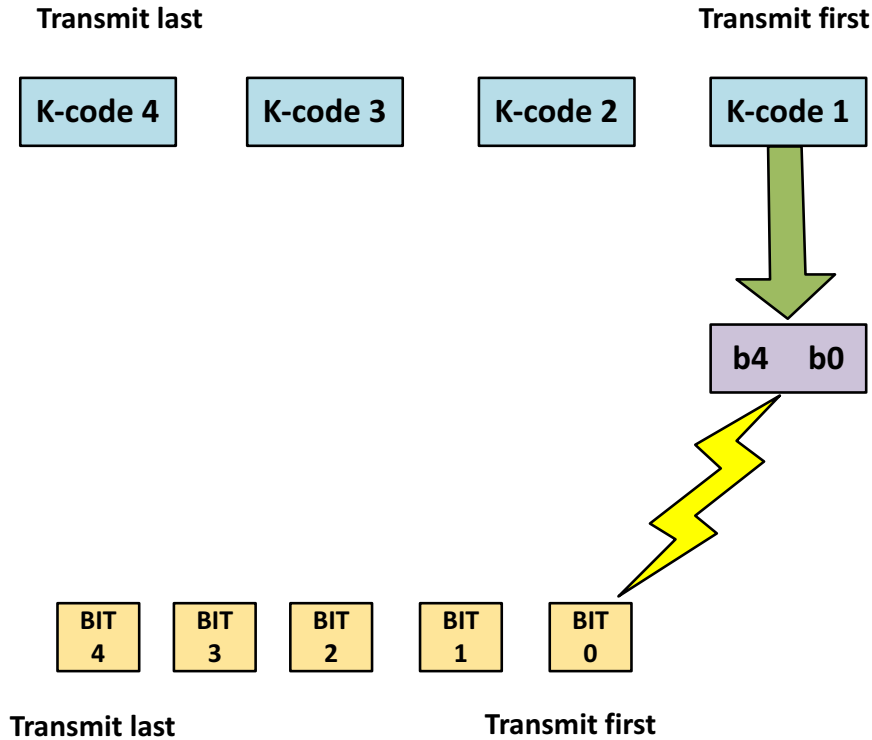
Name	4b	5b Symbol	Description
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
7	0111	01111	hex data 7
8	1000	10010	hex data 8
9	1001	10011	hex data 9
A	1010	10110	hex data A
B	1011	10111	hex data B
C	1100	11010	hex data C
D	1101	11011	hex data D
E	1110	11100	hex data E
F	1111	11101	hex data F
<i>Sync-1</i>	<i>K-code</i>	11000	Startsynch #1
<i>Sync-2</i>	<i>K-code</i>	10001	Startsynch #2
<i>RST-1</i>	<i>K-code</i>	00111	<i>Hard Reset</i> #1
<i>RST-2</i>	<i>K-code</i>	11001	<i>Hard Reset</i> #2
<i>EOP</i>	<i>K-code</i>	01101	<i>EOP</i> End of Packet
	Error	00000	Shall Not be used
	Error	00001	Shall Not be used
	Error	00010	Shall Not be used
	Error	00011	Shall Not be used
	Error	00100	Shall Not be used
	Error	00101	Shall Not be used
<i>Sync-3</i>	<i>K-code</i>	00110	Startsynch #3
	Error	01000	Shall Not be used
	Error	01100	Shall Not be used
	Error	10000	Shall Not be used
	Error	11111	Shall Not be used

5.4 Ordered Sets

Ordered sets **shall** be interpreted according to [Figure 5.1, "Interpretation of ordered sets"](#).

An ordered set consists of 4 *K-codes* sent as shown in [Figure 5.1, "Interpretation of ordered sets"](#).

Figure 5.1 Interpretation of ordered sets



A list of the ordered sets used by USB Power Delivery can be seen in [Table 5.2, "Ordered Sets"](#). *SOP** is a generic term used in place of *SOP/SOP'/SOP''*.

Table 5.2 Ordered Sets

Ordered Set	Reference
<i>Cable Reset</i>	Section 5.6.5, "Cable Reset"
<i>Hard Reset</i>	Section 5.6.4, "Hard Reset"
<i>SOP</i>	Section 5.6.1.2.1, "Start of Packet Sequence (SOP)"
<i>SOP'</i>	Section 5.6.1.2.2, "Start of Packet Sequence Prime (SOP')"
<i>SOP'_Debug</i>	Section 5.6.1.2.4, "Start of Packet Sequence Prime Debug (SOP'_Debug)"
<i>SOP''</i>	Section 5.6.1.2.3, "Start of Packet Sequence Double Prime (SOP'')"
<i>SOP''_Debug</i>	Section 5.6.1.2.5, "Start of Packet Sequence Double Prime Debug (SOP''_Debug)"

The receiver **shall** search for all four *K-codes*. When the receiver finds all four *K-codes* in the correct place, it **shall** interpret this as a **valid** ordered set. When the receiver finds three out of four *K-codes* in the correct place, it **may**

interpret this as a **Valid** ordered set. The receiver **Should** ensure that all four *K-codes* are **Valid** to avoid ambiguity in detection (see [Table 5.3, "Validation of Ordered Sets"](#)).

Table 5.3 Validation of Ordered Sets

	1st code	2nd code	3rd code	4th code
Valid ¹	Corrupt	<i>K-code</i>	<i>K-code</i>	<i>K-code</i>
Valid ¹	<i>K-code</i>	Corrupt	<i>K-code</i>	<i>K-code</i>
Valid ¹	<i>K-code</i>	<i>K-code</i>	Corrupt	<i>K-code</i>
Valid ¹	<i>K-code</i>	<i>K-code</i>	<i>K-code</i>	Corrupt
Valid ² (perfect)	<i>K-code</i>	<i>K-code</i>	<i>K-code</i>	<i>K-code</i>
Invalid (example)	<i>K-code</i>	Corrupt	<i>K-code</i>	Corrupt
1)	May be interpreted as a Valid ordered set.			
2)	Shall be interpreted as a Valid ordered set.			

5.5 Transmitted Bit Ordering

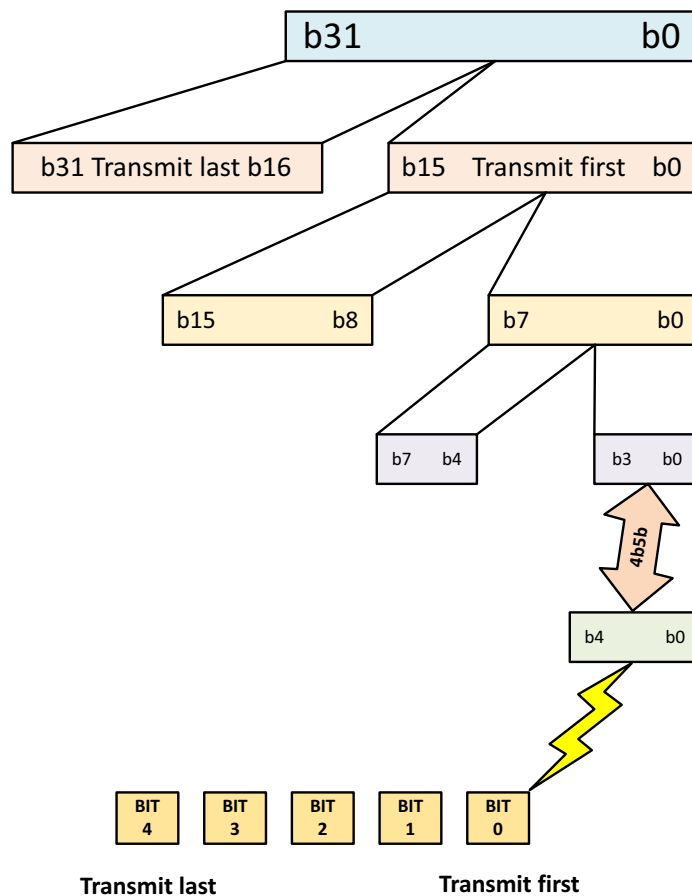
This section describes the order of bits on the wire that **shall** be used when transmitting data of varying sizes. [Table 5.4, "Data Size"](#) shows the different data sizes that are possible.

[Figure 5.2, "Transmit Order for Various Sizes of Data"](#) shows the transmission order that **shall** be followed.

Table 5.4 Data Size

	Unencoded	Encoded
Byte	8-bits	10-bits
Word	16-bits	20- bits
DWord	32-bits	40-bits

Figure 5.2 Transmit Order for Various Sizes of Data



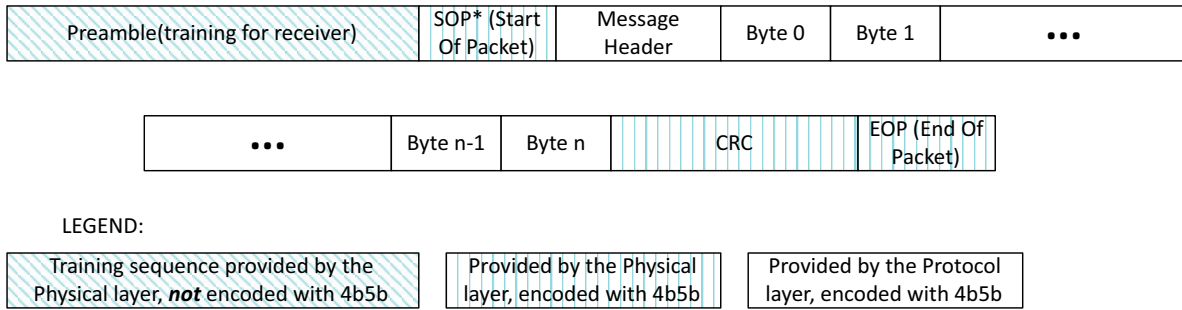
5.6 Packet Format

The *Packet* format **shall** consist of a *Preamble*, an *SOP**, (see [Section 5.6.1.2, "Start of Packet Sequences"](#)), *Packet* data including the *Message Header*, a *CRC* and an *EOP* (see [Section 5.6.1.5, "End of Packet \(EOP\)"](#)). The *Packet* format is shown in [Figure 5.3, "USB Power Delivery Packet Format"](#) and indicates which parts of the *Packet* **shall** be 4b/5b encoded. Once 4b/5b encoded, the entire *Packet* **shall** be transmitted using *BMC* over *CC*.

Note: All the bits in the *Packet*, including the *Preamble*, are *BMC* encoded.

See [Section 6.2.1, "Message Construction"](#) for more details of the *Packet* construction for *Control Messages*, *Data Messages* and *Extended Messages*.

Figure 5.3 USB Power Delivery Packet Format



5.6.1 Packet Framing

The transmission starts with a *Preamble* that is used to allow the receiver to lock onto the carrier. It is followed by a *SOP** (*Start of Packet*). The *Packet* is terminated with an *EOP* (*End of Packet*) *K-code*.

5.6.1.1 Preamble

The *Preamble* is used to achieve lock in the receiver by presenting an alternating series of "0s" and "1s", so the average frequency is the carrier frequency. Unlike the rest of the *Packet*, the *Preamble* **shall** Not be 4b/5b encoded.

The *Preamble* **shall** consist of a 64-bit sequence of alternating 0s and 1s. The *Preamble* **shall** start with a "0" and **shall** end with a "1".

5.6.1.2 Start of Packet Sequences

5.6.1.2.1 Start of Packet Sequence (SOP)

SOP is an ordered set. The *SOP* ordered set is defined as: three *Sync-1* *K-codes* followed by one *Sync-2* *K-code* (see [Table 5.5, "SOP Ordered Set"](#)).

Table 5.5 SOP Ordered Set

K-Code Number	K-Code in Code Table
1	<i>Sync-1</i>
2	<i>Sync-1</i>
3	<i>Sync-1</i>
4	<i>Sync-2</i>

A Power Delivery Capable *Source* or *Sink* **shall** be able to detect and communicate with *Packets* using *SOP*. If a *Valid SOP* is not detected (see [Table 5.3, "Validation of Ordered Sets"](#)) then the whole transmission **shall** be **Discarded**.

Sending and receiving of *SOP* *Packets* **shall** be limited to *PD Capable Ports* on *PDUSB Hosts* and *PDUSB Devices*. *Cable Plugs* and *VPDs* **shall** neither send nor receive *SOP* *Packets*.

Note: PDUSB Devices, even if they have the physical form of a cable (e.g., AMAs), are still required to respond to SOP Packets.

5.6.1.2.2 Start of Packet Sequence Prime (SOP')

The **SOP'** ordered set is defined as: two **Sync-1** K-codes followed by two **Sync-3** K-codes (see [Table 5.6, "SOP' Ordered Set"](#)).

Table 5.6 SOP' Ordered Set

K-Code Number	K-Code in Code Table
1	Sync-1
2	Sync-1
3	Sync-3
4	Sync-3

A VPD **Shall** have SOP' Communication capability. A VPD and a Cable Plug capable of SOP' Communications **Shall** only detect and communicate with Packets starting with **SOP'**.

A Port needing to communicate with a Cable Plug capable of SOP' Communications, Attached between a Port Pair will be able to communicate using both Packets starting with **SOP'** to communicate with the Cable Plug and starting with **SOP** to communicate with its Port Partner.

For a VPD or a Cable Plug supporting SOP' Communications, if a **Valid SOP'** is not detected (see [Table 5.3, "Validation of Ordered Sets"](#)) then the whole transmission **Shall** be **Discarded**. For a Port supporting SOP' Communications if a **Valid SOP** or **SOP'** is not detected (see [Table 5.3, "Validation of Ordered Sets"](#)) then the whole transmission **Shall** be **Discarded**. When there is no Explicit Contract or an Implicit Contract in place a Sink **Shall Not** send SOP' Packets and **Shall Discard** all Packets starting with **SOP'**.

5.6.1.2.3 Start of Packet Sequence Double Prime (SOP'')

The **SOP''** ordered set is defined as the following sequence of K-codes: **Sync-1, Sync-3, Sync-1, Sync-3** (see [Table 5.7, "SOP'' Ordered Set"](#)).

Table 5.7 SOP'' Ordered Set

K-Code Number	K-Code in Code Table
1	Sync-1
2	Sync-3
3	Sync-1
4	Sync-3

A VPD **Shall Not** have SOP'' Communication capability. A Cable Plug capable of SOP'' Communication, **Shall** have a SOP' Communication capability in the other Cable Plug. No cable **Shall** only support SOP'' Communication. A Cable Plug to which SOP'' Communication is assigned **Shall** only detect and communicate with Packets starting with **SOP''** and **Shall Discard** any other Packets.

A Port needing to communicate with such a Cable Plug, Attached between a Port Pair will be able to communicate using Packets starting with **SOP'** and **SOP''** to communicate with the Cable Plugs and Packets starting with **SOP** to communicate with its Port Partner. A Port which supports SOP'' Communication **Shall** also support SOP' Communication and **Shall** co-ordinate SOP* Communication so as to avoid collisions.

For the Cable Plug supporting SOP'' Communication, if a **Valid SOP''** is not detected (see [Table 5.3, "Validation of Ordered Sets"](#)) then the whole transmission **Shall** be **Discarded**. For the Port if a **Valid SOP*** is not detected (see [Table 5.3, "Validation of Ordered Sets"](#)) then the whole transmission **Shall** be **Discarded**.

5.6.1.2.4 Start of Packet Sequence Prime Debug (SOP' _Debug)

The **SOP' _Debug** ordered set is defined as the following sequence of *K-codes*: **Sync-1**, **RST-2**, **RST-2**, **Sync-3** (see [Table 5.8, "SOP' _Debug Ordered Set"](#)). The usage of this Ordered Set is presently undefined.

Table 5.8 SOP' _Debug Ordered Set

K-Code Number	K-Code in Code Table
1	Sync-1
2	RST-2
3	RST-2
4	Sync-3

5.6.1.2.5 Start of Packet Sequence Double Prime Debug (SOP" _Debug)

The **SOP" _Debug** ordered set is defined as the following sequence of *K-codes*: **Sync-1**, **RST-2**, **Sync-3**, **Sync-2** (see [Table 5.9, "SOP" _Debug Ordered Set"](#)). The usage of this Ordered Set is presently undefined.

Table 5.9 SOP" _Debug Ordered Set

K-Code Number	K-Code in Code Table
1	Sync-1
2	RST-2
3	Sync-3
4	Sync-2

5.6.1.3 Packet Payload

The *Packet* data is delivered from the *Protocol Layer* (see [Section 6.2, "Messages"](#)) and **Shall** be encoded with the hex data codes from [Table 5.1, "4b5b Symbol Encoding"](#).

5.6.1.4 CRC

The **CRC** **Shall** be inserted just after the *Payload*. It is described in [Section 5.6.2, "CRC"](#).

5.6.1.5 End of Packet (EOP)

The end of *Packet* marker **Shall** be a single **EOP** *K-code* as defined in [Figure 5.1, "Interpretation of ordered sets"](#). This **Shall** mark the end of the **CRC**. After the **EOP**, the **CRC**-residual **Shall** be checked. If the **CRC** is not good, the whole transmission **Shall** be **Discarded**, if it is good, the *Packet* **Shall** be delivered to the *Protocol Layer*.

Note: An **EOP** **May** be used to prematurely terminate a *Packet* e.g., before sending **Hard Reset** Signaling.

5.6.2 CRC

The *Message Header* and data **Shall** be protected by a 32-bit **CRC**.

- **CRC**-32 protects the data integrity of the data *Payload*. **CRC**-32 is defined as follows:
- The **CRC**-32 polynomial **Shall** be = 04C1_1DB7h.
- The **CRC**-32 Initial value **Shall** be = FFFF_FFFFh.
- **CRC**-32 **Shall** be calculated for all bytes of the *Payload* not inclusive of any *Packet* framing symbols (i.e., excludes the *Preamble*, *SOP**, *EOP*).
- **CRC**-32 calculation **Shall** begin at byte 0, bit 0 and continue to bit 7 of each of the bytes of the *Packet*.
- The remainder of **CRC**-32 **Shall** be complemented.
- The residual of **CRC**-32 **Shall** be C704 DD7Bh.

This inversion of the *CRC*-32 remainder adds an offset of FFFF_FFFFh that will create a constant *CRC*-32 residual of C704_DD7Bh at the receiver side.

Note: The *CRC* implementation is identical to the one used in [\[USB 3.2\]](#).

[Figure 5.4, "CRC-32 Generation"](#) is an illustration of *CRC*-32 generation. The output bit ordering **shall** be as detailed in [Table 5.10, "CRC-32 Mapping"](#).

Figure 5.4 CRC-32 Generation

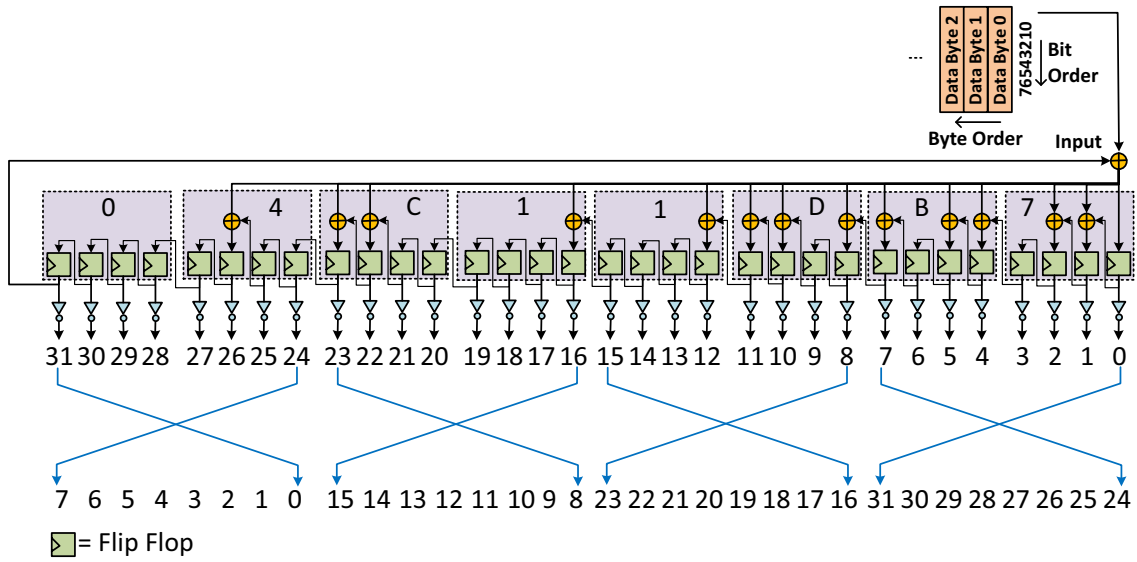


Table 5.10 CRC-32 Mapping

CRC-32	Result Bit Position in CRC-32 Field
0	31
1	30
2	29
3	28
4	27
5	26
6	25
7	24
8	23
9	22
10	21
11	20
12	19
13	18
14	17
15	16
16	15
17	14
18	13
19	12
20	11
21	10
22	9
23	8
24	7
25	6
26	5
27	4
28	3
29	2
30	1
31	0

The CRC-32 **Shall** be encoded before transmission.

5.6.3 Packet Detection Errors

CRC errors, or errors detected while decoding encoded symbols using the code table, **Shall** be treated the same way; the *Message* **Shall** be **Discarded** and a **GoodCRC** Message **Shall Not** be returned.

While the receiver is processing a *Packet*, if at any time the CC-line becomes Idle the receiver **Shall** stop processing the *Packet* and **Discard** it (no **GoodCRC** Message is returned). See [Section 5.8.6.1, "Definition of Idle"](#) for the definition of BMC Idle.

5.6.4 Hard Reset

Hard Reset Signaling is an ordered set of bytes sent with the purpose to be recognized by the PHY Layer. The **Hard Reset** Signaling ordered set is defined as: three **RST-1** K-codes followed by one **RST-2** K-code (see [Table 5.11, "Hard Reset Ordered Set"](#)).

Table 5.11 Hard Reset Ordered Set

K-Code Number	K-Code in Code Table
1	RST-1
2	RST-1
3	RST-1
4	RST-2

A device **Shall** perform a *Hard Reset* when it receives **Hard Reset** Signaling. After receiving the **Hard Reset** Signaling, the device **Shall** reset as described in [Section 6.8.3, "Hard Reset"](#). If a **Valid** *Hard Reset* is not detected (see [Table 5.3, "Validation of Ordered Sets"](#)) then the whole transmission **Shall** be **Discarded**.

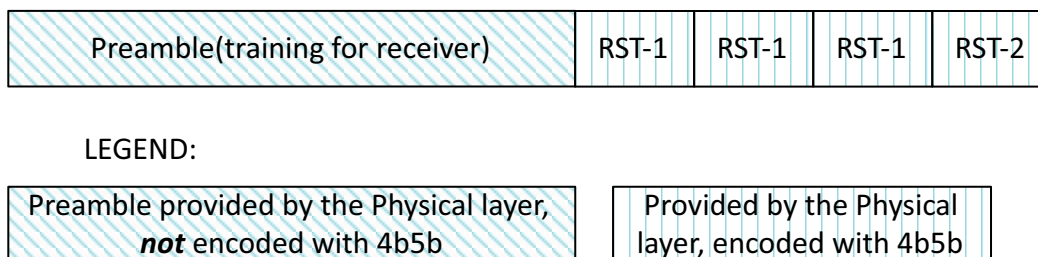
A *Cable Plug* **Shall** perform a *Hard Reset* when it detects **Hard Reset** Signaling being sent between the Port Partners. After receiving the **Hard Reset** Signaling, the device **Shall** reset as described in [Section 6.8.3, "Hard Reset"](#).

The procedure for sending **Hard Reset** Signaling **Shall** be as follows:

- If the PHY Layer is currently sending a *Message*, the *Message* **Shall** be interrupted by sending an **EOP** K-code and the rest of the *Message* **Discarded**.
- If CC is not Idle, wait for it to become Idle (see [Section 5.8.6.1, "Definition of Idle"](#)).
- Wait **tInterFrameGap**.
- If CC is still Idle send the *Preamble* followed by the 4 K-codes for **Hard Reset** Signaling.
- Disable the channel (i.e., stop sending and receiving), reset the PHY Layer and inform the Protocol Layer that the PHY Layer has been reset.
- Re-enable the channel when requested by the Protocol Layer.

[Figure 5.5, "Line format of Hard Reset"](#) shows the line format of **Hard Reset** Signaling which is a *Preamble* followed by the **Hard Reset** Ordered Set.

Figure 5.5 Line format of Hard Reset



5.6.5 Cable Reset

Cable Reset Signaling is an ordered set of bytes sent with the purpose to be recognized by the *PHY* Layer. The **Cable Reset** Signaling ordered set is defined as the following sequence of *K*-codes: **RST-1**, **Sync-1**, **RST-1**, **Sync-3** (see [Table 5.12, "Cable Reset Ordered Set"](#)).

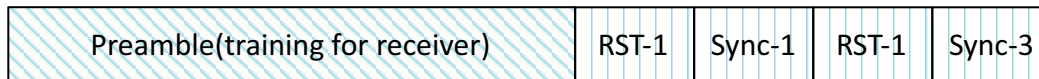
Table 5.12 Cable Reset Ordered Set

K-Code Number	K-Code in Code Table
1	RST-1
2	Sync-1
3	RST-1
4	Sync-3

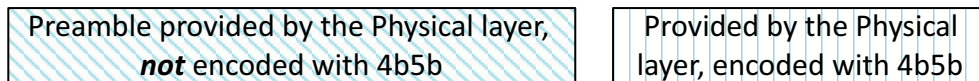
Cable Reset Signaling **Shall** only be sent by the *DFP*. The **Cable Reset** Ordered Set is used to reset the *Cable Plugs* without the need to *Hard Reset* the *Port Partners*. The state of the *Cable Plug* after the **Cable Reset** Signaling **Shall** be equivalent to power cycling the *Cable Plug*.

[Figure 5.6, "Line format of Cable Reset"](#) shows the line format of **Cable Reset** Signaling which is a *Preamble* followed by the **Cable Reset** Ordered Set.

Figure 5.6 Line format of Cable Reset



LEGEND:



5.7 Collision Avoidance

The *PHY Layer* **Shall** monitor the channel for data transmission and only initiate transmissions when *CC* is *Idle*. If the bus *Idle* condition is present, it **Shall** be considered safe to start a transmission provided the conditions detailed in [Section 5.8.5.4, "Inter-Frame Gap"](#) are met. The bus *Idle* condition **Shall** be checked immediately prior to transmission. If transmission cannot be initiated, then the *Packet* **Shall** be **Discarded**. If the *Packet* is **Discarded** because *CC* is not *Idle*, the *PHY Layer* **Shall** signal to the *Protocol Layer* that it has **Discarded** the *Message* as soon as *CC* becomes *Idle*. See [Section 5.8.6.1, "Definition of Idle"](#) for the definition of *Idle CC*.

In addition, during an *Explicit Contract*, the *PHY Layer* **Shall** control the R_p resistor value to avoid collisions between *Source* and *Sink* transmissions. The *Source* **Shall** set an R_p value corresponding to a current of 3A (**SinkTxOK**) to indicate to the *Sink* that it **May** initiate an AMS. The *Source* **Shall** set an R_p value corresponding to a current of 1.5A (**SinkTxNG**) this **Shall** indicate to the *Sink* that it **Shall Not** initiate an AMS and **Shall** only respond to *Messages* as part of an AMS. See [\[USB Type-C 2.4\] \(USB Type-C\)](#) for details of the corresponding R_p values. During the *Implicit Contract* that precedes an *Explicit Contract* (including *Power Role Swap* and *Fast Role Swap*) the R_p resistor value is used to specify *USB Type-C* current and is not used for *Collision Avoidance*.

[Table 5.13, "R_p values used for Collision Avoidance"](#) details the R_p values that **Shall** be used by the *Source* to control *Sink* initiation of an AMS.

Table 5.13 R_p values used for Collision Avoidance

Source R_p	Parameter	Description	Sink Operation	Source Operation
1.5A@5V	SinkTxNG	<i>Sink</i> Transmit "No Go,"	The <i>Sink</i> Shall Not initiate an AMS once $t_{SinkDelay}$ has elapsed after SinkTxNG is asserted.	<i>Source</i> can initiate an AMS t_{SinkTx} after setting R_p to this value.
3A@5V	SinkTxOK	<i>Sink</i> Transmit "Ok"	<i>Sink</i> can initiate an AMS.	<i>Source</i> cannot initiate an AMS while it has this value set.

See also [Section 6.6.16, "Collision Avoidance Timers"](#) and [Section 6.10, "Collision Avoidance"](#).

5.8 Bi-phase Mark Coding (BMC) Signaling Scheme

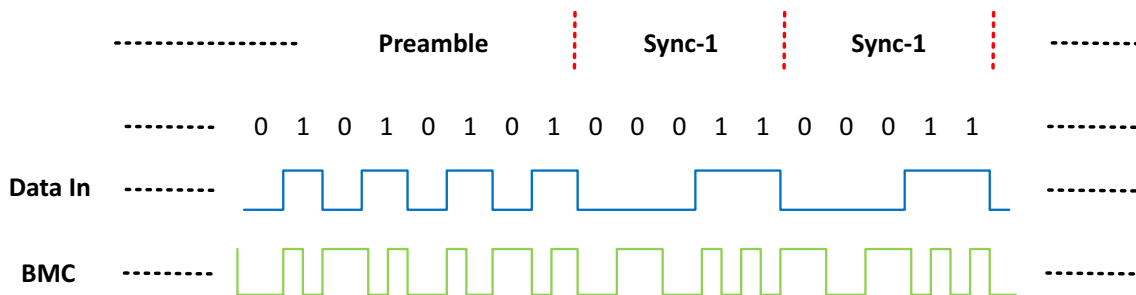
Bi-phase Mark Coding (BMC) is the *PHY Layer Signaling Scheme* for carrying USB Power Delivery Messages. This encoding assumes a dedicated DC connection, over the *CC* wire, which is used for sending PD Messages.

Bi-phase Mark Coding is a version of Manchester coding (see [IEC 60958-1]). In *BMC*, there is a transition at the start of every bit time (*UI*) and there is a second transition in the middle of the *UI* when a 1 is transmitted. *BMC* is effectively DC balanced, (each 1 is DC balanced and two successive zeros are DC balanced, regardless of the number of intervening 1's). It has bounded disparity (limited to 1 bit over an arbitrary *Packet*, so a very low DC level).

[Figure 5.7, "BMC Example"](#) illustrates *Bi-phase Mark Coding*. This example shows the transition from a *Preamble* to the *Sync-1* K-codes of the *SOP* Ordered Set at the start of a *Message*.

Note: Other K-codes can occur after the *Preamble* for Signaling such as *Hard Reset* and *Cable Reset*.

Figure 5.7 BMC Example



5.8.1 Encoding and signaling

BMC uses DC coupled baseband Signaling on *CC*. [Figure 5.8, "BMC Transmitter Block Diagram"](#) shows a block diagram for a Transmitter and [Figure 5.9, "BMC Receiver Block Diagram"](#) shows a block diagram for the corresponding Receiver.

Figure 5.8 BMC Transmitter Block Diagram

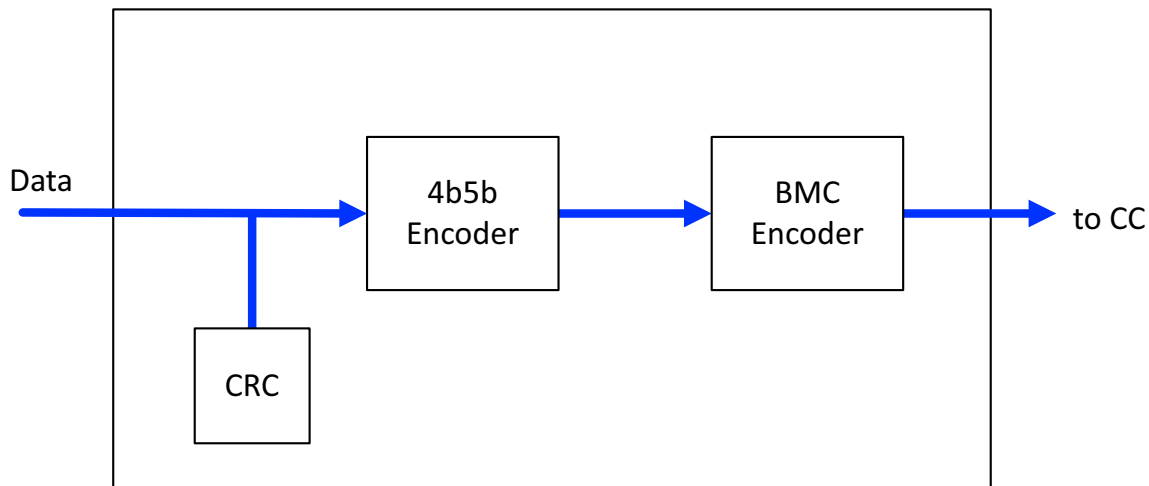
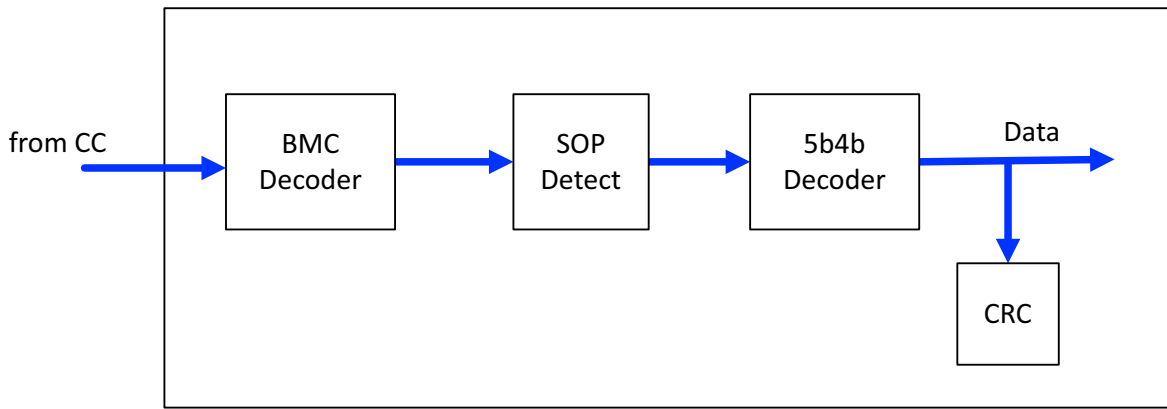


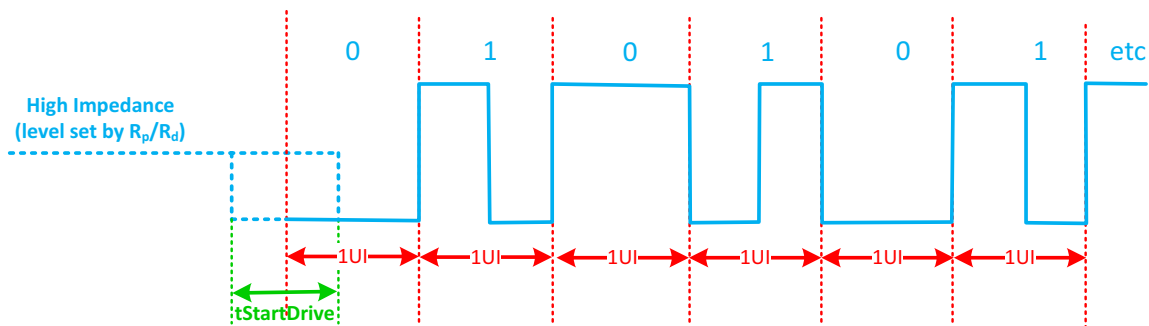
Figure 5.9 BMC Receiver Block Diagram



The USB PD baseband signal **shall** be driven on the CC wire with a tristate driver that **shall** cause a **vSwing** swing on CC. The tristate driver is slew rate limited (see min rise/fall time in [Section 5.8.5, "BMC Transmitter Specifications"](#)) to limit coupling to D+/D- and to other signal lines in the *USB Type-C* fully featured cables (see [\[USB Type-C 2.4\]](#)). This slew rate limiting can be performed either with driver design or an RC filter on the driver output.

When sending the *Preamble*, the transmitter **shall** start by transmitting a low level. The receiver **shall** tolerate the loss of the first edge. The transmitter **may** vary the start of the *Preamble* by **tStartDrive** min (see [Figure 5.10, "BMC Encoded Start of Preamble"](#)).

Figure 5.10 BMC Encoded Start of Preamble



The transmitter **shall** terminate the final bit of the *Frame* by an edge (the "trailing edge") to help ensure that the receiver clocks the final bit. If the trailing edge results in the transmitter driving CC low (i.e., the final half-UI of the *Frame* is high, see [Figure 5.11, "Transmitting or Receiving BMC Encoded Frame Terminated by Zero with High-to-Low Last Transition"](#) and [Figure 5.12, "Transmitting or Receiving BMC Encoded Frame Terminated by One with High-to-Low Last Transition"](#)), then the transmitter:

- **shall** continue to drive CC low for **tHoldLowBMC**.
- **should** release CC to high impedance as soon as possible after min **tHoldLowBMC** and **shall** release CC by max **tEndDriveBMC**.

[Figure 5.11, "Transmitting or Receiving BMC Encoded Frame Terminated by Zero with High-to-Low Last Transition"](#) illustrates the end of a BMC encoded *Frame* with an encoded zero for which the final bit of the *Frame* is terminated by a high to low transition. [Figure 5.12, "Transmitting or Receiving BMC Encoded Frame Terminated by One with High-to-Low Last Transition"](#) illustrates the end of a BMC Encoded *Frame* with an encoded one for which the final bit of the *Frame* is terminated by a high to low transition. Both figures also illustrate the **tInterFrameGap** timing requirement before the start of the next *Frame* when the *Port* has either been transmitting or receiving the previous *Frame* (see [Section 5.8.5.4, "Inter-Frame Gap"](#)).

Figure 5.11 Transmitting or Receiving BMC Encoded Frame Terminated by Zero with High-to-Low Last Transition

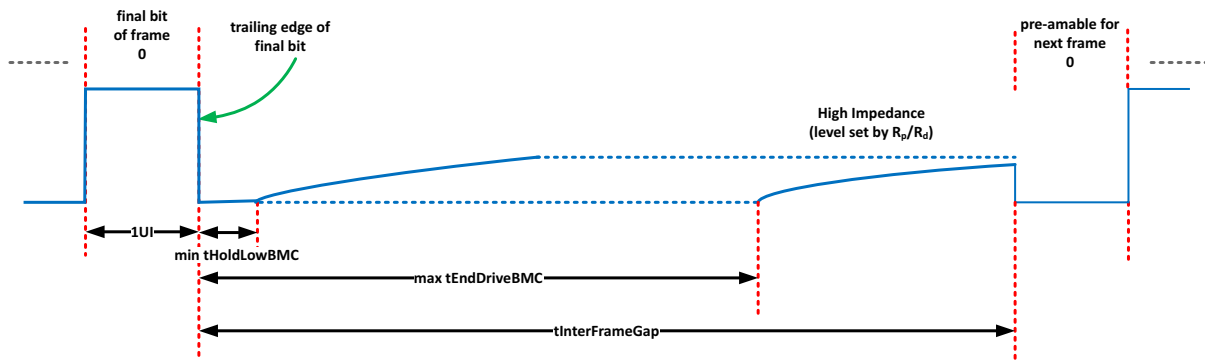
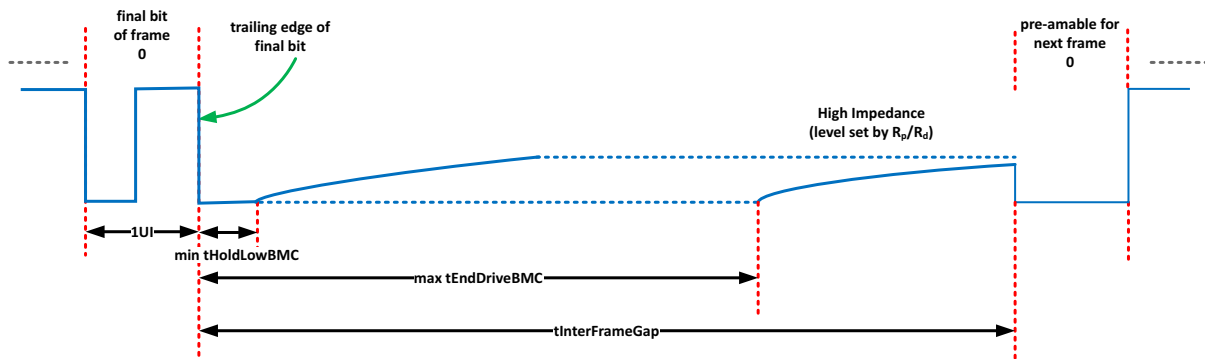


Figure 5.12 Transmitting or Receiving BMC Encoded Frame Terminated by One with High-to-Low Last Transition



If the trailing edge results in the transmitter driving CC high (i.e., the final half-UI of the Frame is low, see [Figure 5.13, "Transmitting or Receiving BMC Encoded Frame Terminated by Zero with Low to High Last Transition"](#) and [Figure 5.14, "Transmitting or Receiving BMC Encoded Frame Terminated by One with Low to High Last Transition"](#)), then the transmitter:

- **Shall** continue to drive CC high for 1 UI.
 - Then **Shall** drive CC low for $tHoldLowBMC$.
- **Should** release CC to high impedance as soon as possible after min $tHoldLowBMC$ and **Shall** release CC by max $tEndDriveBMC$.

[Figure 5.13, "Transmitting or Receiving BMC Encoded Frame Terminated by Zero with Low to High Last Transition"](#) illustrates the ending of a BMC encoded Frame that ends with an encoded zero for which the final bit of the Frame is terminated by a low to high transition. [Figure 5.14, "Transmitting or Receiving BMC Encoded Frame Terminated by One with Low to High Last Transition"](#) illustrates the ending of a BMC encoded Frame that ends with an encoded one for which the final bit of the Frame is terminated by a low to high transition. Both figures also illustrate the $tInterFrameGap$ timing requirement before the start of the next Frame when the Port has either been transmitting or receiving the previous Frame (see [Section 5.8.5.4, "Inter-Frame Gap"](#)).

Figure 5.13 Transmitting or Receiving BMC Encoded Frame Terminated by Zero with Low to High Last Transition

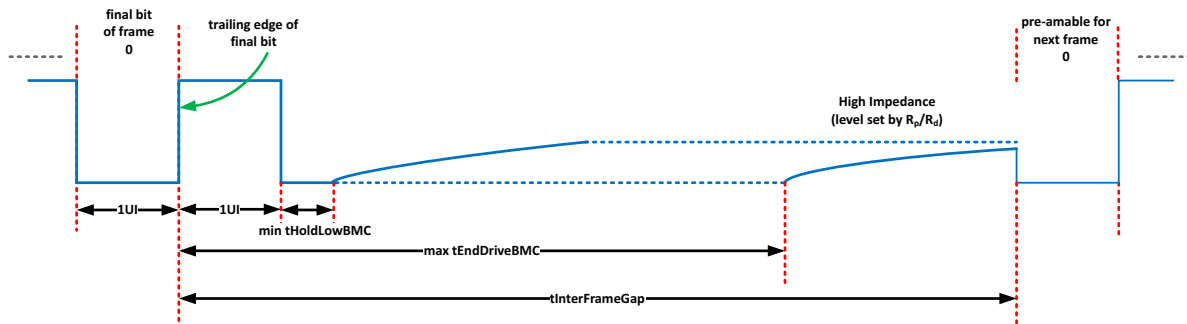
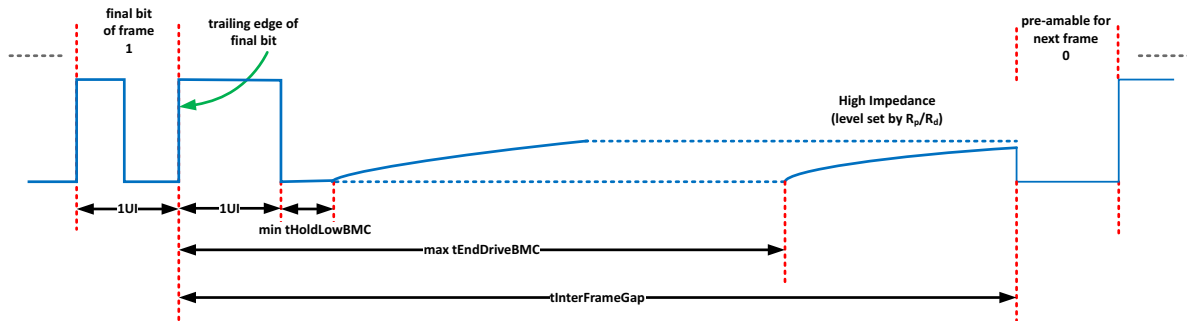


Figure 5.14 Transmitting or Receiving BMC Encoded Frame Terminated by One with Low to High Last Transition



Note: There is no requirement to maintain a timing phase relationship between back-to-back *Packets*.

5.8.2 Transmit and Receive Masks

5.8.2.1 Transmit Masks

The transmitted signal **Shall Not** violate the masks defined in [Figure 5.15, "BMC Tx 'ONE' Mask"](#), [Figure 5.16, "BMC Tx 'ZERO' Mask"](#), [Table 5.14, "BMC Tx Mask Definition, X Values"](#) and [Table 5.15, "BMC Tx Mask Definition, Y Values"](#) at the output of a load equivalent to the cable model and receiver load model described in [Section 5.8.3, "Transmitter Load Model"](#). The masks apply to the full range of R_p/R_d values as defined in [\[USB Type-C 2.4\]](#).

Note: The measurement of the transmitter does not need to accommodate a change in signal offset due to the ground offset when current is flowing in the cable.

The transmitted signal **Shall** have a rise time no faster than t_{Rise} . The transmitted signal **Shall** have a fall time no faster than t_{Fall} . The maximum limits on the rise and fall times are enforced by the Tx inner masks.

Figure 5.15 BMC Tx 'ONE' Mask

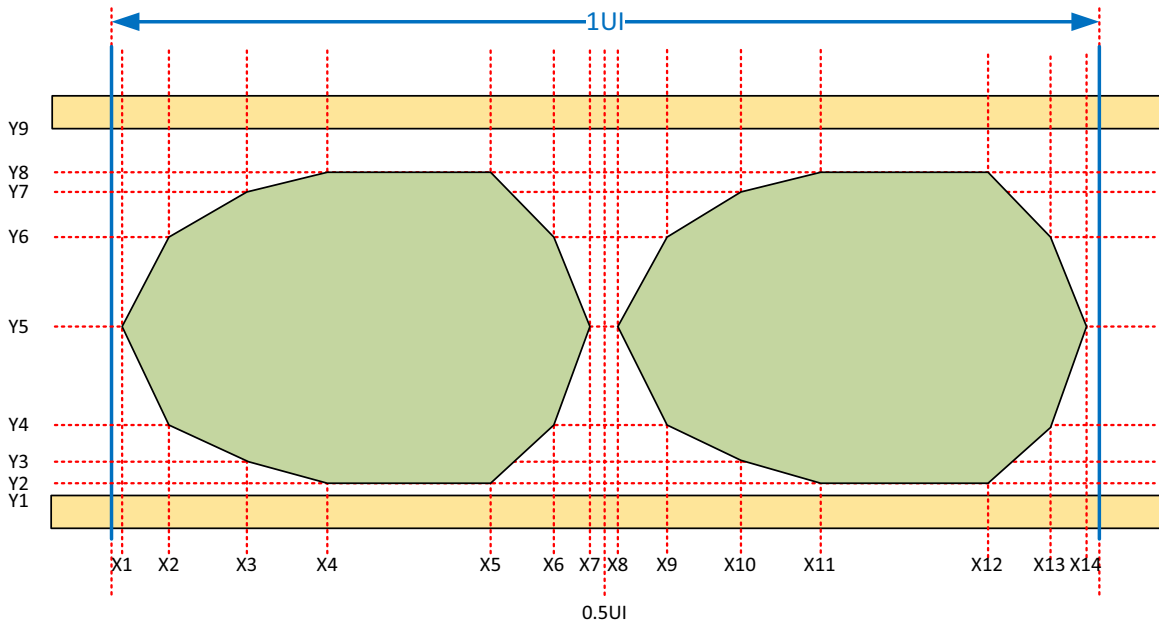


Figure 5.16 BMC Tx 'ZERO' Mask

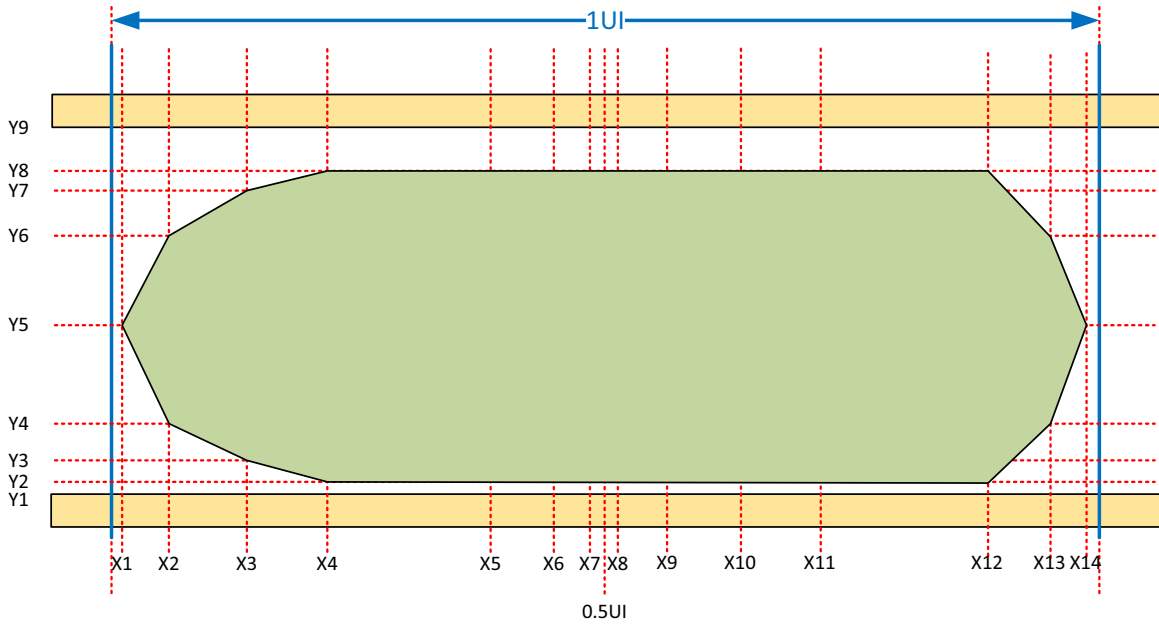


Table 5.14 BMC Tx Mask Definition, X Values

Name	Description	Value	Units
X1Tx	Left Edge of Mask	0.015	UI
X2Tx	see figure	0.07	UI
X3Tx	see figure	0.15	UI
X4Tx	see figure	0.25	UI
X5Tx	see figure	0.35	UI
X6Tx	see figure	0.43	UI
X7Tx	see figure	0.485	UI
X8Tx	see figure	0.515	UI
X9Tx	see figure	0.57	UI
X10Tx	see figure	0.65	UI
X11Tx	see figure	0.75	UI
X12Tx	see figure	0.85	UI
X13Tx	see figure	0.93	UI
X14Tx	Right Edge of Mask	0.985	UI

Table 5.15 BMC Tx Mask Definition, Y Values

Name	Description	Value	Units
Y1Tx	Lower bound of Outer mask	-0.075	V
Y2Tx	Lower bound of inner mask	0.075	V
Y3Tx	see figure	0.15	V
Y4Tx	see figure	0.325	V
Y5Tx	Inner mask vertical midpoint	0.5625	V
Y6Tx	see figure	0.8	V
Y7Tx	see figure	0.975	V
Y8Tx	see figure	1.04	V
Y9Tx	Upper Bound of Outer mask	1.2	V

5.8.2.2 Receive Masks

A *Source* using the *BMC Signaling Scheme* **shall** be capable of receiving a signal that complies with the mask when sourcing power as defined in [Figure 5.17, "BMC Rx 'ONE' Mask when Sourcing Power"](#), [Figure 5.18, "BMC Rx 'ZERO' Mask when Sourcing Power"](#) and [Table 5.16, "BMC Rx Mask Definition"](#). The *Source* Rx mask is bounded by sweeping a Tx mask compliant signal, with added **vNoiseActive** between power neutral and *Source* offsets.

A *Consumer* using the *BMC Signaling Scheme* **shall** be capable of receiving a signal that complies with the mask when sinking power as defined in [Figure 5.21, "BMC Rx 'ONE' Mask when Sinking Power"](#), [Figure 5.22, "BMC Rx 'ZERO' Mask when Sinking Power"](#) and [Table 5.16, "BMC Rx Mask Definition"](#). The *Consumer* Rx mask is bounded by sweeping a Tx mask compliant signal, with added **vNoiseActive** between power neutral and *Consumer* offsets.

Every product using the *BMC Signaling Scheme* **shall** be capable of receiving a signal that complies with the mask when power neutral as defined in [Figure 5.19, "BMC Rx 'ONE' Mask when Power neutral"](#), [Figure 5.20, "BMC Rx 'ZERO' Mask when Power neutral"](#) and [Table 5.16, "BMC Rx Mask Definition"](#).

Dual-Role Power Devices **shall** meet the receiver requirements for a *Source* when providing power during any transmission using the *BMC Signaling Scheme* or a *Sink* when consuming power during any transmission using the *BMC Signaling Scheme*.

Cable Plugs **shall** meet the receiver requirements for both a *Source* and a *Sink* during any transmission using the *BMC Signaling Scheme*.

The parameters used in the masks are specified to be appropriate to either edge triggered or oversampling receiver implementations.

The masks are defined for 'ONE' and 'ZERO' separately as *BMC* enforces a transition at the midpoint of the unit interval while a 'ONE' is transmitted.

The Rx masks are defined to bound the Rx noise after the Rx bandwidth limiting filter with the time constant **tRxFilter** has been applied.

The boundaries of Rx outer mask, **Y1Rx** and **Y5Rx**, are specified according to **vSwing** max and accommodate half of **vNoiseActive** from cable noise coupling and the signal offset **vIRDropGNDC** due to the ground offset when current is flowing in the cable.

The vertical dimension of the Rx inner mask, **Y4Rx** - **Y2Rx**, for power neutral is derived by reducing the vertical dimension of the Tx inner mask, **Y7Tx** - **Y3Tx**, at time location **X3Tx** by **vNoiseActive** to account for cable noise coupling. The received signal is composed of a waveform compliant to the Tx mask plus **vNoiseActive**.

The vertical dimension of the Rx inner mask for sourcing power is derived by reducing the vertical dimension of the Tx inner mask by **vNoiseActive** and **vIRDropGNDC** to account for both cable noise coupling and signal DC offset.

The received signal is composed of a waveform compliant to the Tx mask plus the maximum value of $v_{NoiseActive}$ plus $v_{IRDropGNDC}$ where the $v_{IRDropGNDC}$ value transitions between the minimum and the maximum values as allowed in this spec.

The vertical dimension of the Rx inner mask for sinking power is derived by reducing the vertical dimension of the Tx inner mask by $v_{NoiseActive}$ max and $v_{IRDropGNDC}$ max for account for both cable noise coupling and signal DC offset. The received signal is composed of a waveform compliant to the Tx mask plus the maximum value of $v_{NoiseActive}$ plus $v_{IRDropGNDC}$ where the $v_{IRDropGNDC}$ value transitions between the minimum and the maximum values as allowed in this spec.

The center line of the Rx inner mask, $Y3_{Rx}$, is at half of the nominal v_{Swing} for power neutral, and is shifted up by half of $v_{IRDropGNDC}$ max for sourcing power and is shifted down by half of $v_{IRDropGNDC}$ max for sinking power.

The receiver sensitivity **shall** be set such that the receiver does not treat noise on an undriven signal path as an incoming signal. Signal amplitudes below $v_{NoiseIdle}$ max **shall** be treated as noise when BMC is *Idle*.

Figure 5.17 BMC Rx 'ONE' Mask when Sourcing Power

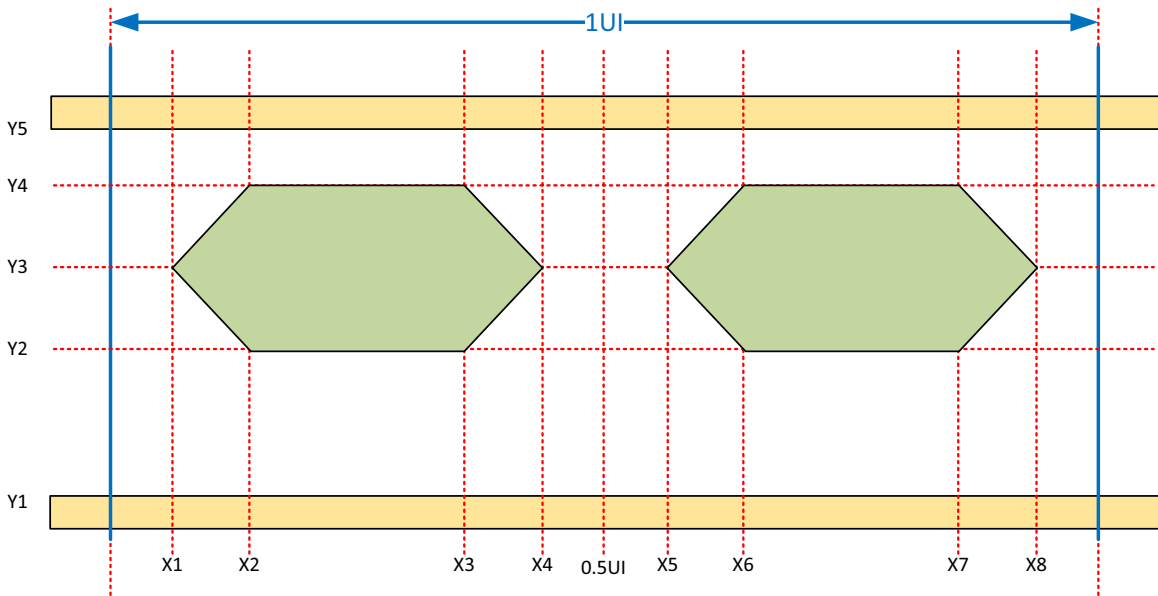


Figure 5.18 BMC Rx 'ZERO' Mask when Sourcing Power

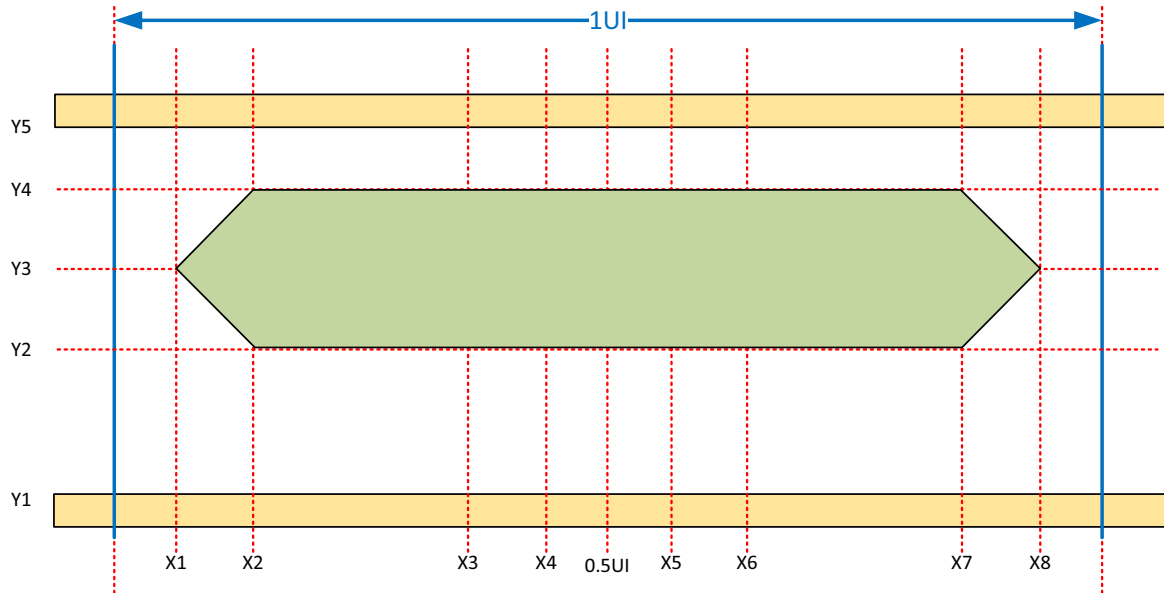


Figure 5.19 BMC Rx 'ONE' Mask when Power neutral

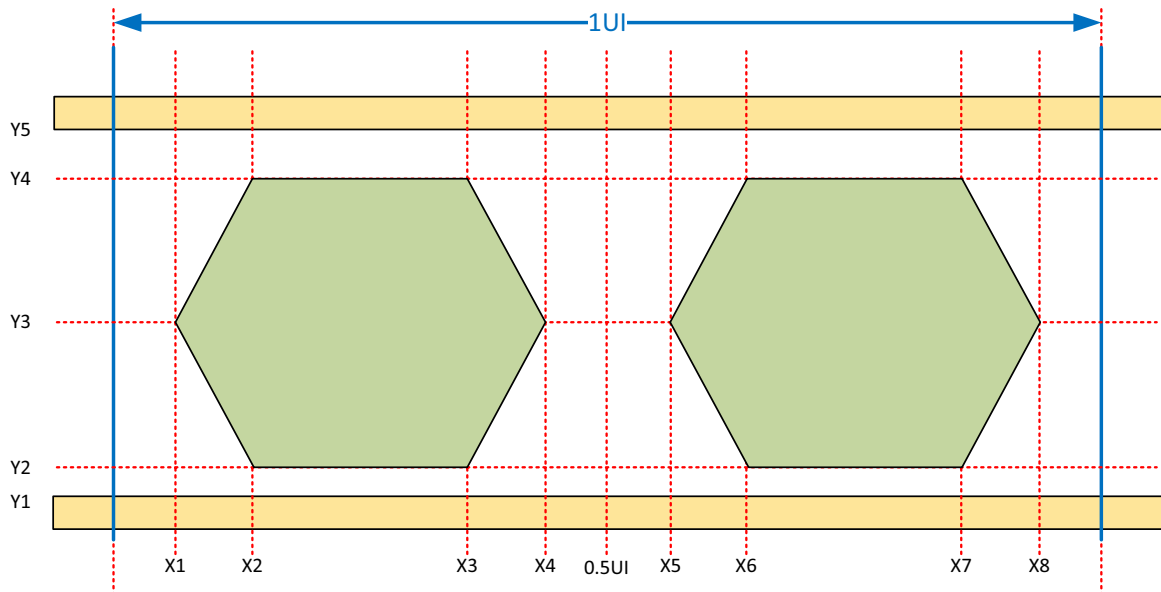


Figure 5.20 BMC Rx 'ZERO' Mask when Power neutral

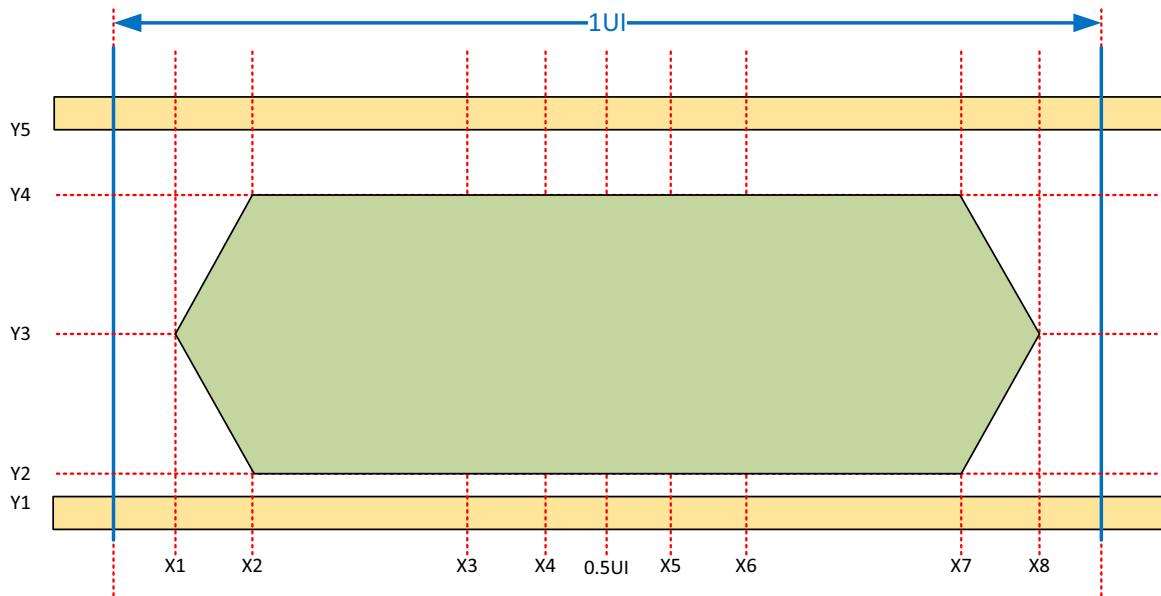


Figure 5.21 BMC Rx 'ONE' Mask when Sinking Power

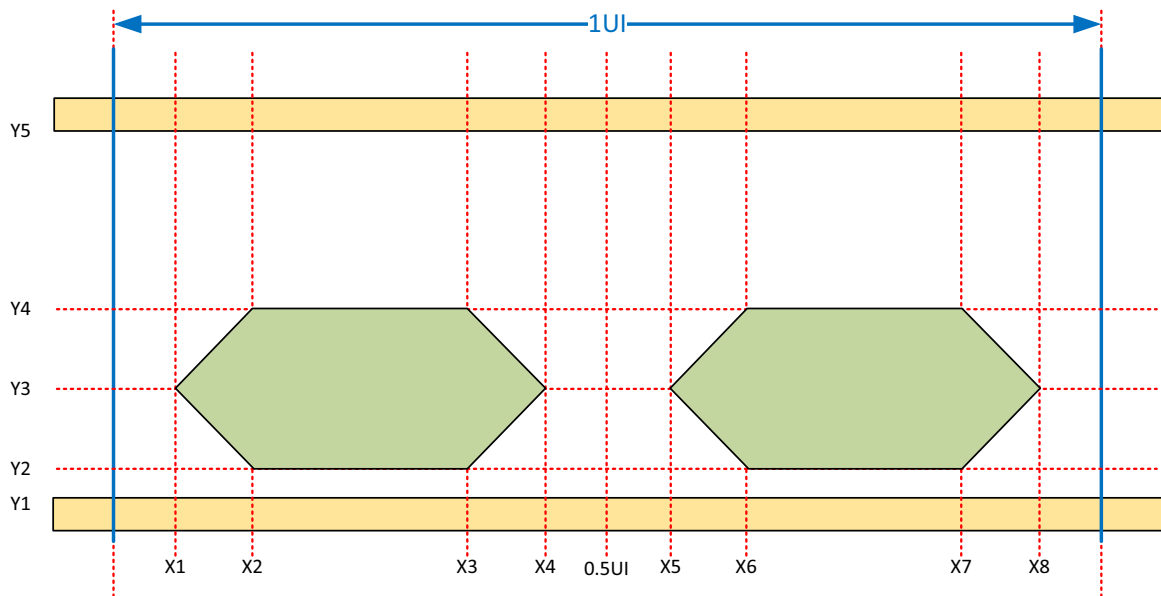


Figure 5.22 BMC Rx 'ZERO' Mask when Sinking Power

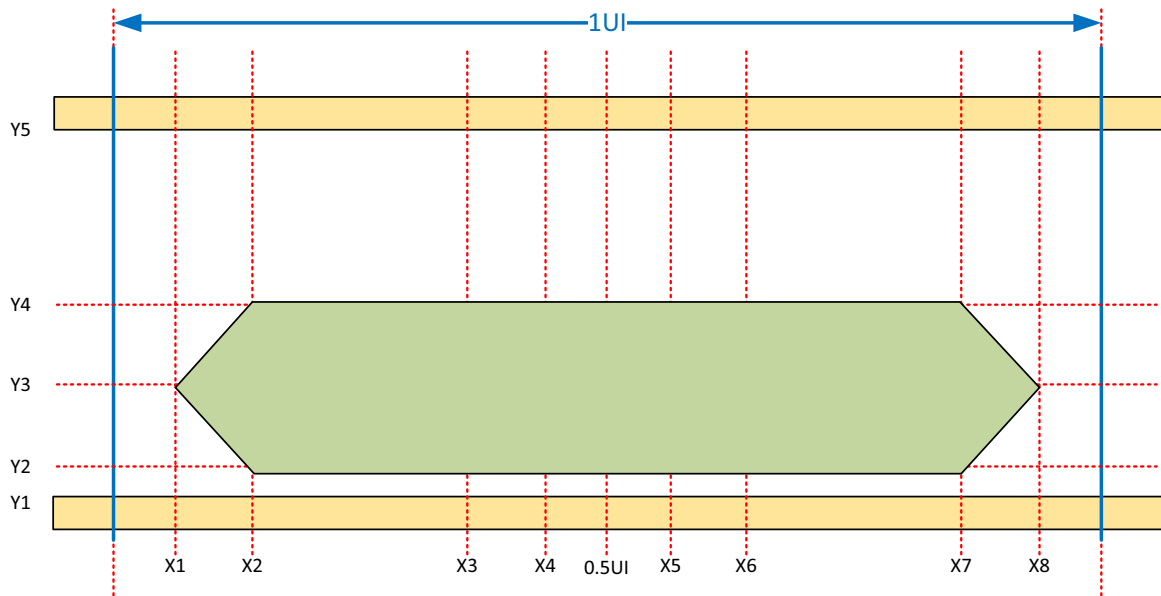


Table 5.16 BMC Rx Mask Definition

Name	Description	Value	Units
X1Rx	Left Edge of Mask	0.07	UI
X2Rx	Top Edge of Mask	0.15	UI
X3Rx	See figure	0.35	UI
X4Rx	See figure	0.43	UI
X5Rx	See figure	0.57	UI
X6Rx	See figure	0.65	UI
X7Rx	See figure	0.85	UI
X8Rx	See figure	0.93	UI
Y1Rx	Lower bound of Outer Mask	-0.3325	V
Y2Rx	Lower Bound of Inner Mask	Y3Rx – 0.205 when sourcing power ¹ or sinking power ¹ . Y3Rx – 0.33 when power neutral ¹ .	V
Y3Rx	Center line of Inner Mask	0.6875 Sourcing Power ¹ . 0.5625 Power Neutral ¹ . 0.4375 Sinking Power ¹ .	V
Y4Rx	Upper bound of Inner mask	Y3Rx + 0.205 when sourcing power ¹ or sinking power ¹ . Y3Rx + 0.33 when power neutral ¹ .	V
Y5Rx	Upper bound of the Outer mask	1.5325	V

1) The position of the center line of the Inner Mask is dependent on whether the receiver is Sourcing or Sinking power or is Power Neutral (see earlier in this section).

5.8.3 Transmitter Load Model

The transmitter load model **shall** be equivalent to the circuit outlined in [Figure 5.23, "Transmitter Load Model for BMC Tx from a Source"](#) for a *Source* and [Figure 5.24, "Transmitter Load Model for BMC Tx from a Sink"](#) for a *Sink*. It is formed by the concatenation of a cable load model and a receiver load model. See [\[USB Type-C 2.4\]](#) for details of the R_p and R_d resistors.

Note: The parameters $zCable_CC$, $tCableDelay_CC$ and $cCablePlug_CC$ are defined in [\[USB Type-C 2.4\]](#).

Figure 5.23 Transmitter Load Model for BMC Tx from a Source

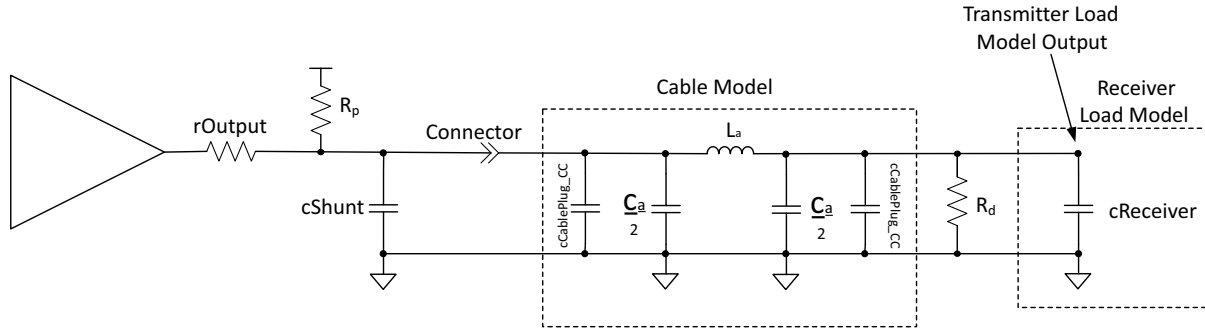
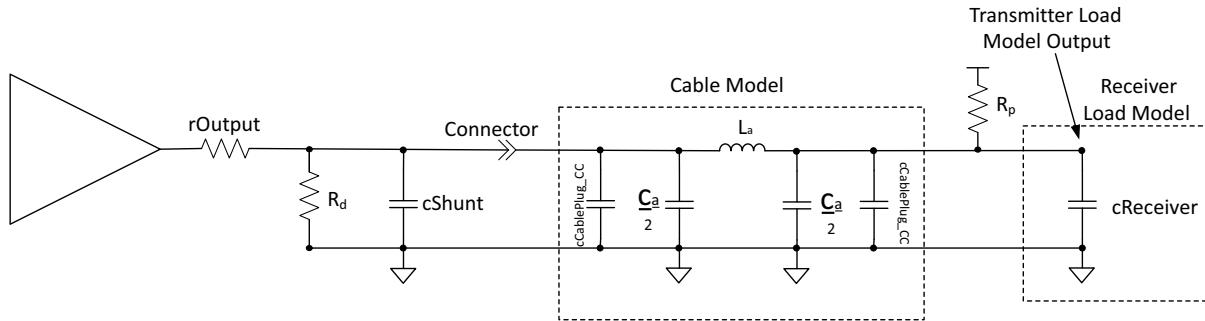


Figure 5.24 Transmitter Load Model for BMC Tx from a Sink



The transmitter system components $rOutput$ and $cShunt$ are illustrated for **Informative** purposes, and do not form part of the transmitter load model. See [Section 5.8.5, "BMC Transmitter Specifications"](#) for a description of the transmitter system design.

The value of the modeled cable inductance, L_a , (in nH) **shall** be calculated from the following formula:

$$L_a = tCableDelay_CC_{max} * zCable_CC_{min}$$

$tCableDelay_CC$ is the modeled signal propagation delay through the cable, and $zCable_CC$ is the modeled cable impedance.

The modeled cable inductance is 640nH for a cable with $zCable_CC_{min} = 32\Omega$ and $tCableDelay_CC_{max} = 20ns$.

The value of the modeled cable capacitance, C_a , (in pF) **shall** be calculated from the following formula:

$$C_a = tCableDelay_CC_{max} / zCable_CC_{min}$$

The modeled cable capacitance is $C_a = 625pF$ for a cable with $zCable_CC_{min} = 32\Omega$ and $tCableDelay_CC_{max} = 20ns$. Therefore, $C_a/2 = 312.5pF$.

$cCablePlug_CC$ models the capacitance of the plug at each end of the cable. $cReceiver$ models the capacitance of the receiver. The maximum values **shall** be used in each case.

Note: The transmitter load model assumes that there are no other return currents on the ground path.

5.8.4 BMC Common specifications

This section defines the common receiver and transmitter requirements.

5.8.4.1 BMC Common Parameters

The electrical requirements specified in [Table 5.17, "BMC Common Normative Requirements"](#) **shall** apply to both the transmitter and receiver.

Table 5.17 BMC Common Normative Requirements

Name	Description	Min	Nom	Max	Units	Comment
<i>fBitRate</i>	Bit rate	270	300	330	Kbps	
<i>tUnitInterval</i>	Unit Interval ¹	3.03		3.70	μs	$1/f\textit{BitRate}$
1) Denotes the time to transmit an unencoded data bit, not the shortest high or low times on the wire after encoding with <i>BMC</i> . A single data bit cell has duration of $1UI$, but a data bit cell with value 1 will contain a centrally placed 01 or 10 transition in addition to the transition at the start of the cell.						

5.8.5 BMC Transmitter Specifications

The transmitter **shall** meet the specifications defined in [Table 5.18, "BMC Transmitter Normative Requirements"](#).

Table 5.18 BMC Transmitter Normative Requirements

Name	Description	Min	Nom	Max	Units	Comment
pBitRate	Maximum difference between the bit-rate during the part of the <i>Packet</i> following the <i>Preamble</i> and the reference bit-rate.			0.25	%	The reference bit rate is the average bit rate of the last 32 bits of the <i>Preamble</i> .
rFRSwapTx	<i>Fast Role Swap Request</i> transmit driver resistance (excluding cable resistance)			5	Ω	Maximum driver resistance of a <i>Fast Role Swap Request</i> transmitter. Assumes a worst case cable resistance of 15Ω as defined in [USB Type-C 2.4] . Note: Based on this value the maximum combined driver and cable resistance of a <i>Fast Role Swap Request</i> transmitter is 20Ω .
tEndDriveBMC	Time to cease driving the line after the end of the last bit of the <i>Frame</i> .			23	μs	Min value is limited by tHoldLowBMC .
tFall	Fall Time	300			ns	10% and 90% amplitude points, minimum is under an unloaded condition.
tHoldLowBMC	Time to cease driving the line after the final high-to-low transition.	1			μs	Max value is limited by tEndDriveBMC .
tInterFrameGap	Time from the end of last bit of a <i>Frame</i> until the start of the first bit of the next <i>Preamble</i> .	25			μs	
tFRSwapTx	<i>Fast Role Swap Request</i> transmit duration	60		120	μs	<i>Fast Role Swap Request</i> is indicated from the <i>Initial Source</i> to the <i>Initial Sink</i> by driving <i>CC</i> low for this time.
tRise	Rise time	300			ns	10% and 90% amplitude points, minimum is under an unloaded condition.
tStartDrive	Time before the start of the first bit of the <i>Preamble</i> when the transmitter shall start driving the line.	-1		1	μs	
vSwing	Voltage Swing	1.05	1.125	1.2	V	Applies to both no load condition and under the load condition specified in Section 5.8.3, "Transmitter Load Model" .
zDriver	Transmitter output impedance	33		75	Ω	<i>Source</i> output impedance at the Nyquist frequency of [USB 2.0] low speed (750 kHz) while the <i>Source</i> is driving the <i>CC</i> line.

5.8.5.1 Capacitance when not transmitting

cReceiver is the capacitance that a *DFP* or *UFP* **Shall** present on the *CC* line when the *DFP* or *UFP*'s receiver is not transmitting on the line. The transmitter **May** have more capacitance than **cReceiver** while driving the *CC* line, but **Shall** meet the waveform mask requirements. Once transmission is complete, the transmitter **Shall** disengage capacitance in excess of **cReceiver** from the *CC* wire within **tInterFrameGap**.

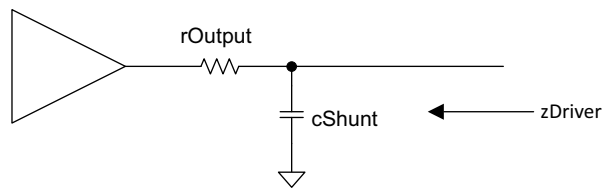
5.8.5.2 Source Output Impedance

Source output impedance **zDriver** is determined by the driver resistance and the shunt capacitance of the *Source* and is hence a frequency dependent term. **zDriver** impacts the noise ingress in the cable. It is specified such that the noise at the Receiver is bounded.

zDriver is defined by the following equation:

$$zDriver = rOutput / (1 + s * rOutput * cShunt)$$

Figure 5.25 Transmitter diagram illustrating zDriver



cShunt **Shall** Not cause a violation of **cReceiver** when not transmitting.

5.8.5.3 Bit Rate Drift

Limits on the drift in **fBitRate** are set to help low-complexity receiver implementations.

fBitRate is the reciprocal of the average bit duration from the previous 32 bits at a given portion of the *Packet*. The change in **fBitRate** during a *Packet* **Shall** be less than **pBitRate**. The reference bit rate (refBitRate) is the average **fBitRate** over the last 32 bits of the *Preamble*. **fBitRate** throughout the *Packet*, including the *EOP*, **Shall** be within **pBitRate** of refBitRate. **pBitRate** is expressed as a percentage:

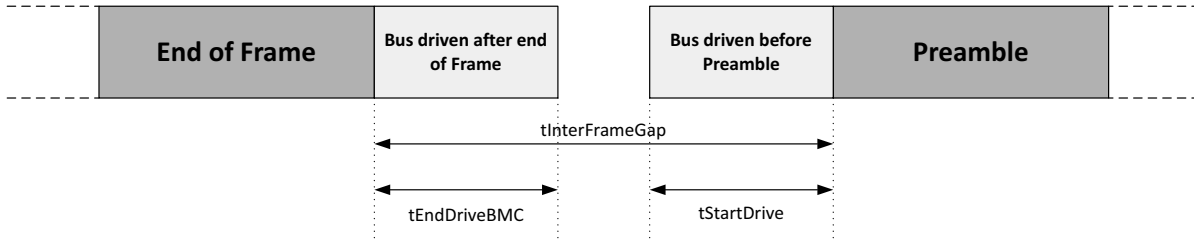
$$pBitRate = | fBitRate - refBitRate | / refBitRate \times 100\%$$

The transmitter **Shall** have the same **pBitRate** for all *Packet* types. The *BIST Carrier Mode* and *Bit Stream* signals are continuous signals without a *Payload*. When checking **pBitRate** any set of 1044 bits (20 bit **SOP** followed by 1024 PRBS bits) within a continuous signal **May** be considered as the part of the *Packet* following the *Preamble* and the 32 preceding bits considered to be the last 32 bits of the *Preamble* used to compute refBitRate.

5.8.5.4 Inter-Frame Gap

Figure 5.26, "Inter-Frame Gap Timings" illustrates the inter-Frame gap timings.

Figure 5.26 Inter-Frame Gap Timings



The transmitter **shall** drive the bus for no longer than $t_{\text{EndDriveBMC}}$ after transmitting the final bit of the *Frame*.

Before starting to transmit the next *Frame*'s *Preamble* the transmitter of the next *Frame* **shall** ensure that it waits for $t_{\text{InterFrameGap}}$ after either:

- Transmitting the previous *Frame*, for example sending the next *Message* in an *AMS* immediately after having sent a **GoodCRC** *Message*, or
- Receiving the previous *Frame*, for example when responding to a received *Message* with a **GoodCRC** *Message*, or
- Observing an *Idle* condition on *CC* (see [Section 5.7, "Collision Avoidance"](#)). In this case the *Port* is waiting to initiate an *AMS* observes *Idle* (see [Section 5.8.6.1, "Definition of Idle"](#)) and then waits $t_{\text{InterFrameGap}}$ before transmitting the *Frame*. See also [Section 5.7, "Collision Avoidance"](#) for details on when an *AMS* can be initiated.

Note: The transmitter is also required to verify a bus *Idle* condition immediately prior to starting transmission of the next *Frame* (see [Section 5.8.6.1, "Definition of Idle"](#)).

The transmitter of the next *Frame* **may** vary the start of the *Preamble* by $t_{\text{StartDrive}}$ (see [Section 5.8.1, "Encoding and signaling"](#)).

See also [Section 5.8.1, "Encoding and signaling"](#) for figures detailing the timings relating to transmitting, receiving, and observing *Idle* in relating to *Frames*.

5.8.5.5 Shorting of Transmitter Output

A Transmitter in a *Port* or *Cable Plug* **shall** tolerate having its output be shorted to ground for t_{FRSwapTx} max. This is due to the potential for *Fast Role Swap* to be signaled while the Transmitter is in the process of transmitting (see [Section 5.8.5.6, "Fast Role Swap Transmission"](#)).

5.8.5.6 Fast Role Swap Transmission

The *Fast Role Swap* process is intended for use by a *PDUSB Hub* that presently has an external supply and is providing power both through its downstream *Ports* to *USB Devices* and upstream to a *USB Host* such as a laptop. On removal of the external wall supply *Fast Role Swap* enables a *VBUS* supply to be maintained by allowing the *USB Host* to apply **vSafe5V** when it sees *VBUS* droop below **vSafe5V** after having detected *Fast Role Swap Signaling*. The *Fast Role Swap AMS* is then used to correctly assign *Source/Sink Power Roles* and configure the R_p/R_d resistors (see [Section 8.3.2.8, "Fast Role Swap"](#)).

The *Initial Source* **Shall** signal a *Fast Role Swap Request* by driving *CC* to ground with a resistance of less than **rFRSwapTx** for **tFRSwapTx**. The *Initial Source* **Shall** only send a *Fast Role Swap Request* when it has an *Explicit Contract*. The *Initial Source* **May** send a *Fast Role Swap Request* even if it has not yet had its *Sink Capabilities* queried by the *Initial Sink*. On transmission of the *Fast Role Swap Request* any pending *Messages* **Shall** be **Discarded** (see [Section 6.12.2.2.1, "Common Protocol Layer Message Transmission State Diagram"](#)).

The *Fast Role Swap Signaling* **May** override any active transmissions.

Since the *Initial Sink*'s response to the *Fast Role Swap* signal is to send an **FR_Swap** Message, the *Initial Source* **Shall** ensure R_p is set to **SinkTxOK** once the *Fast Role Swap Signaling* is complete.

5.8.6 BMC Receiver Specifications

The receiver **shall** meet the specifications defined in [Table 5.19, "BMC Receiver Normative Requirements"](#).

Table 5.19 BMC Receiver Normative Requirements

Name	Description	Min	Nom	Max	Units	Comment
cReceiver	CC receiver capacitance	200		600	pF	The <i>DFP</i> or <i>UFP</i> system shall have capacitance within this range when not transmitting on the line.
nBER	Bit error rate, S/N = 25 dB			10 ⁻⁶		
nTransitionCount	Transitions for signal detect	3				Number of transitions to be detected to declare bus non- <i>Idle</i> .
tFRSwapRx	Fast Role Swap Request detection time	30		50	μs	A <i>Fast Role Swap Request</i> results in the receiver detecting a signal low for at least this amount of time.
tRxFilter	Rx bandwidth limiting filter (digital or analog)	100			ns	Time constant of a single pole filter to limit broad-band noise ingress ¹ .
tTransitionWindow	Time window for detecting non- <i>Idle</i>	12		20	μs	
vFRSwapCableTx	Fast Role Swap Request voltage detection threshold	490	520	550	mV	The <i>Fast Role Swap Request</i> must be below this voltage threshold to be detected.
vIRDropGNDC	Cable Ground <i>IR Drop</i>			250	mV	As specified in [USB Type-C 2.4] .
vNoiseActive	Noise amplitude when <i>BMC</i> is active.			165	mV	Peak-to-peak noise from <i>VBUS</i> , [USB 2.0] and SBU lines after the Rx bandwidth limiting filter with the time constant tRxFilter has been applied.
vNoiseIdle	Noise amplitude when <i>BMC</i> is <i>Idle</i> .			300	mV	Peak-to-peak noise from <i>VBUS</i> , [USB 2.0] and SBU lines after the Rx bandwidth limiting filter with the time constant tRxFilter has been applied.
zBmcRx	Receiver Input Impedance	1			MΩ	

1) Broad-band noise ingress¹ is due to coupling in the cable interconnect.

5.8.6.1 Definition of Idle

BMC Collision Avoidance is performed by the detection of signal transitions at the receiver. Detection is active when **nTransitionCount** transitions occur at the receiver within a time window of **tTransitionWindow**. After waiting **tTransitionWindow** without detecting **nTransitionCount** transitions the bus **shall** be declared *Idle*.

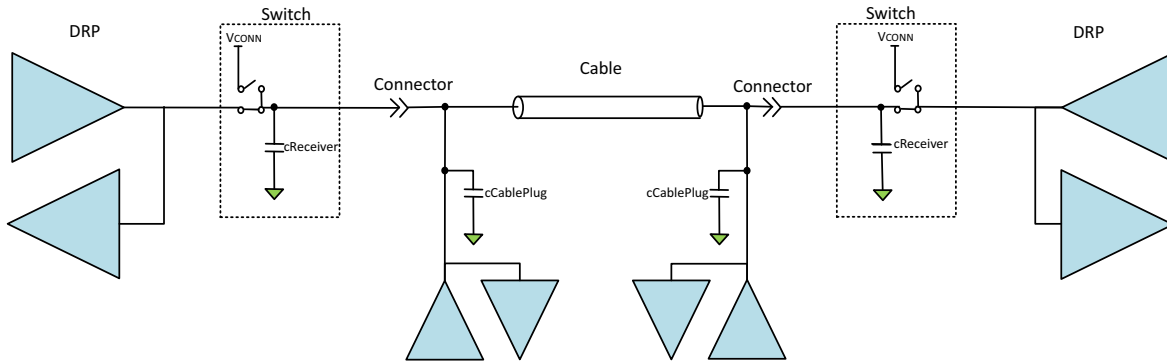
Refer to [Section 5.8.5.4, "Inter-Frame Gap"](#) for details of when transmissions **may** start.

5.8.6.2 Multi-Drop

The *BMC Signaling Scheme* is suitable for use in *Multi-Drop* configurations containing one or two *BMC Multi-Drop* transceivers connected to the *CC* wire, where one or both ends of a cable contains a *Multi-Drop* transceiver. In this specification the location of the *Multi-Drop* transceiver is referred to as the *Cable Plug*.

[Figure 5.27, "Example Multi-Drop Configuration showing two DRPs"](#) below illustrates a typical *Multi-Drop* configuration with two *DRPs*.

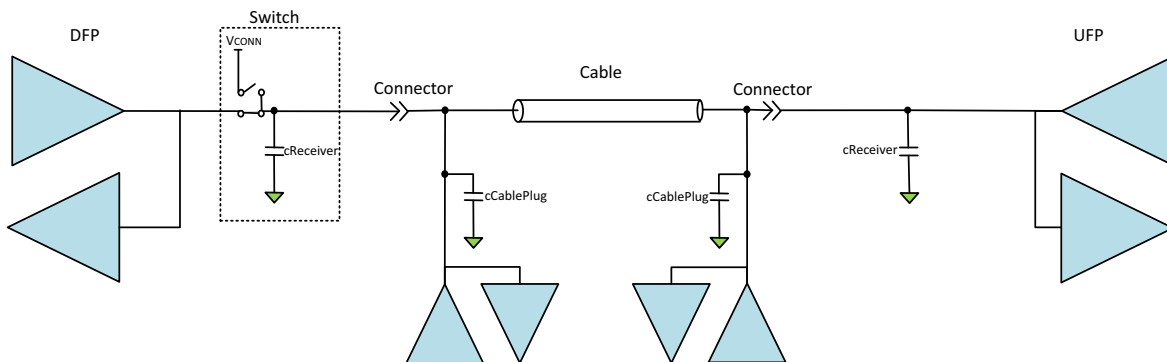
Figure 5.27 Example Multi-Drop Configuration showing two DRPs



The *Multi-Drop* transceiver **shall** obey all the electrical characteristics specified in this section except for those relating to capacitance. The maximum capacitance allowed for the *Multi-Drop* node when not driving the line is *cCablePlug_{CC}* defined in [\[USB Type-C 2.4\]](#). There are no constraints as to the distance of the *Multi-Drop* transceiver from the end of the plug. The *Multi-Drop* transceiver(s) **may** be located anywhere along the cable including the plugs. The *Multi-Drop* transceiver suffers less from ground offset compared to the transceivers in the *USB Host* or *USB Device* and contributes no significant reflections.

It is possible to have a configuration at *Attach* where one *Port* can be a *VCONN Source* and the other *Port* is not able to be a *VCONN Source*, such that there is no switch in the second *Port*. An example of a *DFP* with a switch *Attached* to a *UFP* without a switch is outlined in [Figure 5.28, "Example Multi-Drop Configuration showing a DFP and UFP"](#). The capacitance on the *CC* line for a *Port* not able to be a *VCONN Source* **shall** still be within *cReceiver* except when transmitting.

Figure 5.28 Example Multi-Drop Configuration showing a DFP and UFP



5.8.6.3 Fast Role Swap Detection

An *Initial Sink* prepares for a *Fast Role Swap* by ensuring that once it has detected the *Fast Role Swap Request* its power supply is ready to respond by applying **vSafe5V** according to the timing detailed in [Section 7.1.13, "Fast Role Swap"](#). The *Initial Sink* **shall** only respond to the *Fast Role Swap Request* when all the following conditions have been met:

- An *Explicit Contract* has been established and the *Sink Capabilities* of the *Initial Source* have been received by, and at the request of, the *Initial Sink*.
- The *Sink Capabilities* Message received from the *Initial Source* has at least one of the *Fast Role Swap* bits set in its 5V Fixed Supply PDO.
- The *Initial Sink* is able and willing to source the current requested by the *Initial Source* in the *Fast Role Swap* bits of its *Sink Capabilities* Message.

On detection of the *Fast Role Swap Request* any pending Messages **shall** be **Discarded** (see [Section 6.12.2.2.1, "Common Protocol Layer Message Transmission State Diagram"](#)).

When the *Initial Sink* is prepared for a *Fast Role Swap* and the bus is idle the CC voltage averaged over **tFRSwapRx** min remains above 0.7V (see [\[USB Type-C 2.4\]](#)) since the *Source* R_p is either 1.5A or 3.0A. However, **vNoiseIdle** noise **may** cause the CC line voltage to reach $0.7V - vNoiseIdle/2$ for short durations. When the *Initial Sink* is prepared for a *Fast Role Swap* while it is transmitting and the *Initial Source* is sending a *Fast Role Swap Request*, the transmission will be attenuated such that the peak CC voltage will not exceed **vFRSwapCableTx** min. Therefore, when the *Initial Sink* is prepared for a *Fast Role Swap*, it **shall not** detect a *Fast Role Swap Request* when the CC voltage, averaged over **tFRSwapRx** min, is above 0.7V. When the *Initial Sink* is prepared for a *Fast Role Swap*, it **shall** detect a CC voltage lower than **vFRSwapCableTx** min for **tFRSwapRx** as a *Fast Role Swap Request*.

Note: The *Initial Sink* is not required to average the CC voltage to meet these requirements.

The *Initial Sink* **shall** initiate the *Fast Role Swap* AMS within **tFRSwapInit** of detecting the *Fast Role Swap Request* in order to assign the R_p/R_d resistors to the correct *Ports* and to re-synchronize the state machines (see [Section 6.3.19, "FR Swap Message"](#)).

The *Initial Sink* **shall** become the *New Source* and **shall** start supplying **vSafe5V** at USB Type-C current (see [\[USB Type-C 2.4\]](#)) no later than **tSrcFRSwap** after V_{BUS} has dropped below **vSafe5V**. An *Initial Sink* **shall** disable its V_{BUS} Disconnect Threshold detection circuitry while *Fast Role Swap* detection is active.

Note: While power is transitioning the V_{CONN} *Source* to the *Cable Plug(s)* cannot be guaranteed.

5.9 Built in Self-Test (BIST)

The following sections define *BIST* functionality which *Shall* be supported.

5.9.1 BIST Carrier Mode

In *BIST Carrier Mode*, the *PHY Layer* *Shall* send out a *BMC* encoded continuous string of alternating "1"s and "0"s. This enables the measurement of power supply noise and frequency drift.

Note: This transmission is a purely a sequence of alternating bits and *Shall Not* be formatted as a *Packet*.

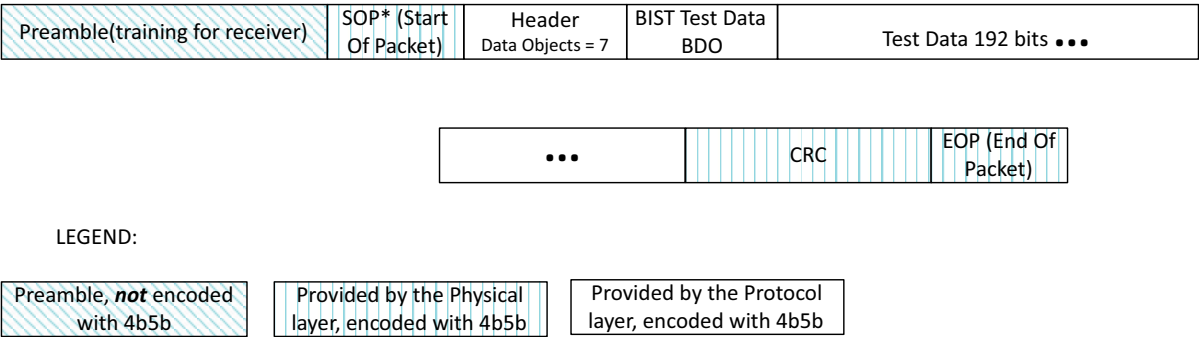
See also [Section 6.4.3, "BIST Message"](#).

5.9.2 BIST Test Data Mode

A *BIST Test Data* Message is used by the *Tester* to send various *Tester* generated *Test Patterns* to the *UUT* in order to test the *UUT*'s receiver. See also [Section 6.4.3, "BIST Message"](#).

[Figure 5.29, "Test Frame"](#) shows the *Test Frame* which *Shall* be sent by the *Tester* to the *UUT*. The *BIST* Message, with a *BIST Test Data* *BIST Data Object* consists of a *Preamble*, followed by *SOP**, followed by the *Message Header* with a data length of 7 *Data Objects*, followed a *BIST Test Data* *BIST Data Object*, followed by 6 *Data Objects* containing test data, followed by the *CRC* and then an *EOP*.

Figure 5.29 Test Frame



6 Protocol Layer

6.1 Overview

This chapter describes the requirements of the USB Power Delivery Specification's *Protocol Layer* including:

- Details of how *Messages* are constructed and used.
- Use of timers and timeout values.
- Use of *Message* and retry counters.
- Reset operation.
- Error handling.
- *State* behavior.

Refer to [Section 2.6, "Architectural Overview"](#) for an overview of the theory of operation of USB Power Delivery.

6.2 Messages

This specification defines three types of *Messages*:

- *Control Messages* that are short and used to manage the *Message* flow between *Port Partners* or to exchange *Messages* that require no additional data. *Control Messages* are 16 bits in length.
- *Data Messages* that are used to exchange information between a pair of *Port Partners*. *Data Messages* range from 48 to 240 bits in length.
 - Some examples of *Data Messages* are:
 - Those used to expose *Capabilities* and *Negotiate* power.
 - Those used for the *BIST*.
 - Those that are *Vendor Defined Messages*.
- *Extended Messages* that are used to exchange information between a pair of *Port Partners*. *Extended Messages* are up to **MaxExtendedMsgLen** bytes.
 - Some examples of *Extended Messages* are:
 - Those used for *Source* and *Battery* information.
 - Those used for Security.
 - Those used for Firmware Update.
 - Those that are *Vendor Defined Extended Messages*.

6.2.1 Message Construction

All *Messages* **shall** be composed of a *Message Header* and a variable length (including zero) data portion. A *Message* either originates in the *Protocol Layer* and is passed to the *PHY Layer*, or it is received by the *PHY Layer* and is passed to the *Protocol Layer*.

[Figure 6.1, "USB Power Delivery Packet Format for a Control Message"](#) illustrates a *Control Message* as part of a *Packet* showing the parts are provided by the *Protocol Layer* and *PHY Layer*.

Figure 6.1 USB Power Delivery Packet Format for a Control Message

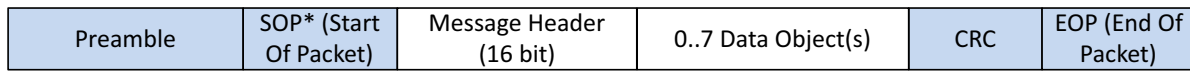
Preamble	SOP* (Start Of Packet)	Message Header (16 bit)	CRC	EOP (End Of Packet)
----------	------------------------	-------------------------	-----	---------------------

Legend:



[Figure 6.2, "USB Power Delivery Packet Format including Data Message Payload"](#) illustrates a *Data Message* as part of a *Packet* showing the parts are provided by the *Protocol Layer* and *PHY Layer*.

Figure 6.2 USB Power Delivery Packet Format including Data Message Payload

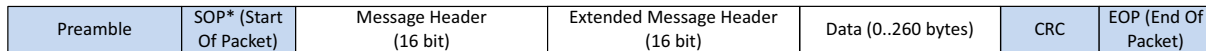


Legend:



[Figure 6.3, "USB Power Delivery Packet Format including an Extended Message Header and Payload"](#) illustrates an *Extended Message* as part of a *Packet* showing the parts are provided by the *Protocol Layer* and *PHY Layer*.

Figure 6.3 USB Power Delivery Packet Format including an Extended Message Header and Payload



Legend:



6.2.1.1 Message Header

Every *Message* **Shall** start with a *Message Header* as shown in:

- [Figure 6.1, "USB Power Delivery Packet Format for a Control Message"](#)
- [Figure 6.2, "USB Power Delivery Packet Format including Data Message Payload"](#)
- [Figure 6.3, "USB Power Delivery Packet Format including an Extended Message Header and Payload"](#)

and as defined in [Table 6.1, "Message Header"](#). The *Message Header* contains basic information about the *Message* and the *PD Port Capabilities*.

The *Message Header* **May** be used standalone as a *Control Message* when the *Number of Data Objects* field is zero or as the first part of a *Data Message* when the *Number of Data Objects* field is non-zero.

Table 6.1 Message Header

Bit(s)	Start of Packet	Field Name	Reference
15	SOP*	<i>Extended</i>	Section 6.2.1.1.1
14...12	SOP*	<i>Number of Data Objects</i>	Section 6.2.1.1.2
11...9	SOP*	<i>MessageID</i>	Section 6.2.1.1.3
8	SOP only	<i>Port Power Role</i>	Section 6.2.1.1.4
	SOP'/SOP''	<i>Cable Plug</i>	Section 6.2.1.1.7
7...6	SOP*	<i>Specification Revision</i>	Section 6.2.1.1.5
5	SOP only	<i>Port Data Role</i>	Section 6.2.1.1.6
	SOP'/SOP''	<i>Reserved</i>	Section 1.4.2
4...0	SOP*	<i>Message Type</i>	Section 6.2.1.1.8

6.2.1.1.1 Extended

The 1-bit *Extended* field **Shall** be set to zero to indicate a *Control Message* or *Data Message* and set to one to indicate an *Extended Message*.

The **Extended** field **Shall** apply to all *SOP* Packet* types.

6.2.1.1.2 Number of Data Objects

When the **Extended** field is set to zero the 3-bit **Number of Data Objects** field **Shall** indicate the number of 32-bit *Data Objects* that follow the *Message Header*. When this field is zero the *Message* is a *Control Message* and when it is non-zero, the *Message* is a *Data Message*.

The **Number of Data Objects** field **Shall** apply to all *SOP* Packet* types.

When both the **Extended** bit and **Chunked** bit are set to one, the **Number of Data Objects** field **Shall** indicate the number of *Data Objects* in the *Message* padded to the 4-byte boundary including the *Extended Message Header* as part of the first *Data Object*.

When the **Extended** bit is set to one and **Chunked** bit is set to zero, the **Number of Data Objects** field **Shall** be **Reserved**.

Note: In this case, the *Message* length is determined solely by the **Data Size** field in the *Extended Message Header*.

6.2.1.1.3 MessageID

The 3-bit **MessageID** field is the value generated by a rolling counter maintained by the originator of the *Message*. The **MessageIDCounter** **Shall** be initialized to zero at power-on as a result of a *Soft Reset*, or a *Hard Reset*. The **MessageIDCounter** **Shall** be incremented when a *Message* is successfully received as indicated by receipt of a **GoodCRC** *Message*.

Note: The usage of **MessageID** during testing with *BIST Messages* is defined in [\[USBPDCompliance\]](#).

The **MessageID** field **Shall** apply to all *SOP* Packet* types.

6.2.1.1.4 Port Power Role

The 1-bit **Port Power Role** field **Shall** indicate the *Port's* present *Power Role*:

- 0b Sink
- 1b Source

Messages, such as **Get_Sink_Cap_Extended**, that are only ever sent by a *Source*, **Shall** always have the **Port Power Role** field set to *Source*. Similarly, *Messages* such as the **Request** *Message* that are only ever sent by a *Sink* **Shall** always have the **Port Power Role** field set to *Sink*.

During the *Power Role Swap AMS*, for the *Initial Source Port*, the **Port Power Role** field **Shall** be set to *Sink* in the **PS_RDY** *Message* indicating that the *Initial Source's* power supply is turned off (see [Table 8.60, "Steps for a Successful Source Initiated Power Role Swap Sequence"](#) and [Table 8.63, "Steps for a Successful Sink Initiated Power Role Swap Sequence"](#)).

During the *Power Role Swap AMS*, for the *Initial Sink*, the **Port Power Role** field **Shall** be set to *Source* for *Messages* initiated by the *Policy Engine* after receiving the **PS_RDY** *Message* from the *Initial Source* (see [Table 8.60, "Steps for a Successful Source Initiated Power Role Swap Sequence"](#) and [Table 8.63, "Steps for a Successful Sink Initiated Power Role Swap Sequence"](#)).

During the *Fast Role Swap AMS*, for the *Initial Source Port*, the **Port Power Role** field **Shall** be set to *Sink* in the **PS_RDY** *Message* indicating that *V_{BUS}* is not being driven by the *Initial Source* and is within **vSafe5V** (see [Figure 8.39, "Successful Fast Role Swap Sequence"](#)).

During the *Fast Role Swap AMS*, for the *Initial Sink Port*, the **Port Power Role** field **Shall** be set to *Source* for *Messages* initiated by the *Policy Engine* after receiving the **PS_RDY** *Message* from the *Initial Source* (see [Figure 8.39, "Successful Fast Role Swap Sequence"](#)).

Note: The **GoodCRC** *Message* sent by the *Initial Sink* in response to the **PS_RDY** *Message* from the *Initial Source* will have its **Port Power Role** field set to *Sink* since this is initiated by the *Protocol Layer*. Subsequent

Messages initiated by the Policy Engine, such as the **PS_RDY** Message sent to indicate that VBUS is ready, will have the **Port Power Role** field set to Source.

The **Port Power Role** field of a received Message **Shall Not** be verified by the receiver and **Shall Not** lead to Soft Reset, Hard Reset or Error Recovery if it is incorrect.

The **Port Power Role** field **Shall** only be defined for SOP Packets.

6.2.1.1.5 Specification Revision

The **Specification Revision** field **Shall** be one of the following values (except 11b):

- 00b - Revision 1.0 (**Deprecated**)
- 01b - Revision 2.0
- 10b - Revision 3.x
- 11b - **Reserved, Shall Not** be used.

To ensure interoperability with existing PDUSB products, PDUSB products **Shall** support every PD Specification Revision starting from **[USB 2.0]** for SOP*; the only exception to this is a VPD which **Shall Ignore** Messages sent with PD Specification Revision 2.0 and earlier.

After a physical or logical (USB Type-C® Error Recovery) Attach, a Port discovers the common Specification Revision level between itself and its Port Partner and/or the Cable Plug(s), and uses this Specification Revision level until a Detach, Hard Reset or Error Recovery happens.

After detection of the Specification Revision to be used, all PD communications **Shall** comply completely with the relevant Revision of the PD specification.

The 2-bit **Specification Revision** field of a **GoodCRC** Message does not carry any meaning and **Shall** be considered as don't care by the recipient of the Message. The sender of a **GoodCRC** Message **Shall** set the **Specification Revision** field to 01b (Revision 2.0) when responding to a Message that contains 01b in the **Specification Revision** field of the Message Header. The sender of a **GoodCRC** Message **May** set the **Specification Revision** field to 01b or 10b when responding to a Message that contains 10b (Revision 3.x) in the **Specification Revision** field of the Message Header.

The **Specification Revision** field **Shall** apply to all SOP* Packet types.

An Attach event or a Hard Reset **Shall** cause the detection of the applicable Specification Revision to be performed for both Ports and Cable Plugs according to the rules stated below:

When the Source Port first communicates with the Sink Port the **Specification Revision** field **Shall** be used as described by the following steps:

- 1) The Source Port sends a **Source Capabilities** Message to the Sink Port setting the **Specification Revision** field to the highest Revision of the Power Delivery Specification the Source Port supports.
- 2) The Sink Port responds with a **Request** Message setting the **Specification Revision** field to the highest Revision of the Power Delivery Specification the Sink Port supports that is equal to or lower than the Specification Revision received from the Source Port.
- 3) The Source and Sink Ports **Shall** use the Specification Revision in the **Request** Message from the Sink in step 2 in all subsequent communications until a Detach, Hard Reset, or Error Recovery happens.

Prior to entering the First Explicit Contract, the VCONN Source **Shall** use the following steps to establish a Specification Revision level:

- 1) The VCONN Source sends a **Discover Identity REQ** to the Cable Plug (SOP) setting the **Specification Revision** field in the Message to the highest Revision of the Power Delivery Specification the VCONN Source supports. After a VCONN Swap the required **Soft Reset** / **Accept** Message exchange is used for the same purpose (see [Section 6.3.13, "Soft Reset Message"](#)).

- 2) The *Cable Plug* responds with a **Discover Identity ACK** setting the **Specification Revision** field in the *Message* to the highest *Revision* of the Power Delivery Specification the *VCONN Source* supports that is equal to or lower than the *Specification Revision* it received from the *Source Port*.
- 3) The *Cable Plug* and *VCONN Source* **Shall** communicate using the lower of the two revisions until an *Explicit Contract* has been established.
- 4) [Table 6.2, "Revision Interoperability during an Explicit Contract"](#) shows the *Specification Revision* that **Shall** be used between the *Port Partners* and the *Cable Plugs* when the *Specification Revision* has been discovered and an *Explicit Contract* is in place.

Notes:

- A *VCONN Source* that does not communicate with the *Cable Plug(s)* **May** skip the above procedure.
- When a *Cable Plug* does not respond to a *Revision 3.x Discover Identity REQ* with a **Discover Identity ACK** or **BUSY** the *VCONN Source* **May** repeat steps 1-4 using a *Revision 2.0 Discover Identity REQ* in step 1 before establishing that there is no *Cable Plug* to communicate with.

A *VCONN Source* that supports *Revision 3.x* of the Power Delivery Specification **May** communicate with a *Cable Plug* also supporting *Revision 3.x* using *Revision 3.x* Compliant Communications regardless of the *Specification Revision* of its *Port Partner* while no *Explicit Contract* exists. After an *Explicit Contract* has been established the *Port Partners* and *Cable Plug(s)* **Shall** use [Table 6.2, "Revision Interoperability during an Explicit Contract"](#) to determine the *Revision* to be used.

All data in all *Messages* **Shall** be consistent with the **Specification Revision** field in the *Message Header* for that particular *Message*.

A *Cable Plug* **Shall Not** save the state of the agreed *Specification Revision*. A *Cable Plug* **Shall** respond with the highest *Specification Revision* it supports that is equal to or lower than the *Specification Revision* contained in the *Message* received from the *VCONN Source*.

Cable Plugs **Shall** operate using the same *Specification Revision* for both *SOP'* and *SOP''*. Cable assemblies with two *Cable Plugs* **Shall** operate using the same *Specification Revision* for both *Cable Plugs*.

See [Table 6.2, "Revision Interoperability during an Explicit Contract"](#) for details of how various *Revisions* **Shall** inter-operate.

Table 6.2 Revision Interoperability during an Explicit Contract

Port 1 Revision	Cable Plug Revision	Port 2 Revision	Port to Port Operating Revision	Port to Cable Plug Operating Revision
2	2	2	2	2
2	2	3	2	2
2	3	2	2	2
2	3	3	2	2
3	2	2	2	2
3	2	3	3	2
3	3	2	2	2
3	3	3	3	3

6.2.1.1.6 Port Data Role

The 1-bit **Port Data Role** field **Shall** indicate the *Port's* present *Data Role*:

- 0b *UFP*
- 1b *DFP*

The **Port Data Role** field **Shall** only be defined for *SOP Packets*. For all other *SOP* Packets* the **Port Data Role** field is **Reserved** and **Shall** be set to zero.

If a *USB Type-C Port* receives a *Message* with the **Port Data Role** field set to the same *Data Role* as its current *Data Role*, except for the **GoodCRC** *Message*, *USB Type-C Error Recovery* actions as defined in [\[USB Type-C 2.4\]](#) **Shall** be performed.

For a *USB Type-C Port* the **Port Data Role** field **Shall** be set to the default value at *Attachment* after a *Hard Reset*: 0b for a *Port* with R_d asserted and 1b for a *Port* with R_p asserted.

In the case that a *Port* is not *USB Communications* capable, at *Attachment* a *Source Port* **Shall** default to *DFP* and a *Sink Port* **Shall** default to *UFP*.

6.2.1.1.7 Cable Plug

The 1-bit **Cable Plug** field **Shall** indicate whether this *Message* originated from a *Cable Plug* or *VPD*:

- 0b *Message* originated from a *DFP* or *UFP*.
- 1b *Message* originated from a *Cable Plug* or *VPD*

The **Cable Plug** field **Shall** only apply to *SOP' Packet* and *SOP'' Packet* types.

6.2.1.1.8 Message Type

The 5-bit **Message Type** field **Shall** indicate the type of *Message* being sent. To fully decode the **Message Type**, the **Number of Data Objects** field is first examined to determine whether the *Message* is a *Control Message* or a *Data Message*. Then the specific **Message Type** can be found in [Table 6.5, "Control Message Types"](#) or [Table 6.6, "Data Message Types"](#).

The **Message Type** field **Shall** apply to all *SOP* Packet* types.

6.2.1.2 Extended Message Header

Extended Messages (indicated by the **Extended** field being set in the *Message Header*) **Shall** contain an *Extended Message Header* following the *Message Header* as shown in [Figure 6.3, "USB Power Delivery Packet Format including an Extended Message Header and Payload"](#) and defined in "[Table 6.3, "Extended Message Header"](#)".

Extended Messages contain *Data Blocks* of **Data Size**, defined in the *Extended Message*, that are either sent in a single *Message* or as a series of *Chunks*. When the *Data Block* is sent as a series of *Chunks*, each *Chunk* in the series, except for the last *Chunk*, **Shall** contain **MaxExtendedMsgChunkLen** bytes. The last *Chunk* in the series **Shall** contain the remainder of the *Data Block* and so could be less than **MaxExtendedMsgChunkLen** bytes and **Shall** be padded to the next 4-byte *Data Object* boundary.

6.2.1.2.1 Chunked

Table 6.3 Extended Message Header

Bit(s)	Start of Packet	Field Name	Reference
15	<i>SOP*</i>	Chunked	Section 6.2.1.2.1
14...11	<i>SOP*</i>	Chunk Number	Section 6.2.1.2.2
10	<i>SOP*</i>	Request Chunk	Section 6.2.1.2.3
9	<i>SOP*</i>	Reserved	
8...0	<i>SOP*</i>	Data Size	Section 6.2.1.2.4

The *Port Partners* **Shall** use the **Unchunked Extended Messages Supported** field in the *Source_Capabilities* *Message* and **Unchunked Extended Messages Supported** field in the *Request* *Message* to determine whether to send *Messages* of **Data Size** > **MaxExtendedMsgLegacyLen** bytes in a single *Unchunked Extended Message* (see [Section 6.4.1.2.1.6, "Unchunked Extended Messages Supported"](#) and [Section 6.4.2.6, "Unchunked Extended Messages Supported"](#)).

When either *Port Partner* only supports *Chunked Extended Messages*:

- The **Chunked** bit in every *Extended Message* **Shall** be set to one.
- Every *Extended Message* of **Data Size** > **MaxExtendedMsgLegacyLen** **Shall** be transmitted between the *Port Partners* in *Chunks*
- The **Number of Data Objects** in the *Message Header* **Shall** indicate the number of *Data Objects* in the *Message* padded to the 4-byte boundary including the *Extended Message Header* as part of the first *Data Object*.

The conditions listed above **Shall** apply until the *Port Pair* is *Detached*, there is a *Hard Reset*, there is *Error Recovery* or the *Source* removes power (except during a *Power Role Swap* or *Fast Role Swap* when the *Initial Source* removes power in order to for the *New Source* to apply power).

When both *Port Partners* support *Unchunked Extended Messages*:

- The **Chunked** bit in every *Extended Message* **Shall** be set to zero.
- Every *Extended Message* **Shall** be transmitted between the *Port Partners* *Unchunked*.
- The **Number of Data Objects** in the *Message Header* is **Reserved**.

The conditions listed above **Shall** apply until the *Port Pair* is *Detached*, there is a *Hard Reset*, there is *Error Recovery* or the *Source* removes power (except during a *Power Role Swap* or *Fast Role Swap* when the *Initial Source* removes power in order to for the *New Source* to apply power).

When sending *Extended Messages* to the *Cable Plug* the *VCONN Source* **Shall** only send *Chunked Extended Messages*. *Cable Plugs* **Shall** always send *Extended Messages* of **Data Size** > **MaxExtendedMsgLegacyLen** *Chunked* and **Shall** set the **Chunked** bit in every *Extended Message* to one.

When *Extended Messages* are supported *Chunking* **Shall** be supported.

6.2.1.2.2 Chunk Number

The **Chunk Number** field **Shall** only be **Valid** in a *Message* if the **Chunked** flag is set to one. If the **Chunked** flag is set to zero the **Chunk Number** field **Shall** also be set to zero.

The **Chunk Number** field is used differently depending on whether the *Message* is a request for *Data*, or a requested *Data Block* being returned:

- In a request for *data* the **Chunk Number** field indicates the number of the *Chunk* being requested. The requester **Shall** only set this field to the number of the next *Chunk* in the series (the next *Chunk* after the last received *Chunk*).
- In the requested *Data Block* the **Chunk Number** field indicates the number of the *Chunk* being returned. The **Chunk Number** for each *Chunk* in the series **Shall** start at zero and **Shall** increment for each *Chunk* by one up to a maximum of 9 corresponding to 10 *Chunks* in total.

6.2.1.2.3 Request Chunk

The **Request Chunk** bit **Shall** only be used for the *Chunked* transfer of an *Extended Message* when the **Chunked** bit is set to 1 (see [Figure 6.7, "Example Security Request sequence Chunked \(Chunked bit = 1\)"](#)). For *Unchunked Extended Message* transfers, *Messages* **Shall** be sent and received without the request/response mechanism (see [Figure 6.4, "Example Security Request sequence Unchunked \(Chunked bit = 0\)"](#)).

The **Request Chunk** bit **Shall** be set to one to indicate that this is a request for a *Chunk* of a *Data Block* and **Shall** be set to zero to indicate that this is a *Chunk* response containing a *Chunk*. Except for *Chunk zero*, a requested *Chunk* of a *Data Block* **Shall** only be returned as a *Chunk* response to a corresponding request for that *Chunk*. Both the *Chunk* request and the *Chunk* response **Shall** contain the same value in the **Message Type** field. When the **Request Chunk** bit is set to one the **Data Size** field **Shall** be zero.

6.2.1.2.4 Data Size

The **Data Size** field **Shall** indicate how many bytes of data in total are in *Data Block* being returned. The total number of data bytes in the *Message* **Shall Not** exceed **MaxExtendedMsgLen**.

If the **Data Size** field is less than **MaxExtendedMsgLegacyLen** and the **Chunked** bit is set then the *Packet Payload* **Shall** be padded to the next 4-byte *Data Object* boundary with zeros (0x00).

If the **Data Size** field is greater than expected for a given *Extended Message* but less than or equal to **MaxExtendedMsgLen** then the expected fields in the *Message* **Shall** be processed appropriately and the additional fields **Shall** be **Ignored**.

6.2.1.2.5 Extended Message Examples

The following examples illustrate the transmission of *Extended Messages* both *Chunked* (**Chunked** bit is one) and *Unchunked* (**Chunked** bit is zero). The examples use a **Security_Request** *Message* of **Data Size** 7 bytes which is responded to by a **Security_Response** *Message* of **Data Size** 30 bytes. The sizes of these *Messages* are arbitrary and are used to illustrate *Message* transmission; they are not intended to correspond to genuine security related *Messages*.

During *Negotiation* of the *Explicit Contract* after connection, the *Port Partners* use the **Unchunked Extended Messages Supported** field in the **Source_Capabilities** *Message* and **Unchunked Extended Messages Supported** field in the **Request** *Message* to determine the value of the **Chunked** bit (see [Table 6.4, "Use of Unchunked Message Supported bit"](#)). When both *Port Partners* support *Unchunked Extended Messages* then the **Chunked** bit is zero otherwise the **Chunked** bit is one.

The **Chunked** bit is used to determine whether:

- The *Chunk* request/response mechanism is used.
- *Extended Messages* are *Chunked*.
- Padding is applied.
- The **Number of Data Objects** field is used.

The following examples illustrate the expected usage in each case.

Table 6.4 Use of Unchunked Message Supported bit

		Source: Source_Capabilities <i>Message</i>	
		<i>Unchunked Message</i> Supported bit = 0	<i>Unchunked Message</i> Supported bit = 1
Sink: Request <i>Message</i>	<i>Unchunked Message</i> Supported bit = 0	Chunked bit = 1	Chunked bit = 1
	<i>Unchunked Message</i> Supported bit = 1	Chunked bit = 1	Chunked bit = 0

6.2.1.2.5.1 Security_Request/Security_Response Unchunked Example

[Figure 6.4, "Example Security_Request sequence Unchunked \(Chunked bit = 0\)"](#) illustrates a typical sequence for a **Security_Request** *Message* responded to by a **Security_Response** *Message* using *Unchunked Extended Messages* (**Chunked** bit is zero) between a *USB Host* and a *Charger*. The entire *Data Block* is returned in one *Message*. The *Chunk* request/response mechanism is not used.

Figure 6.4 Example Security_Request sequence Unchunked (Chunked bit = 0)

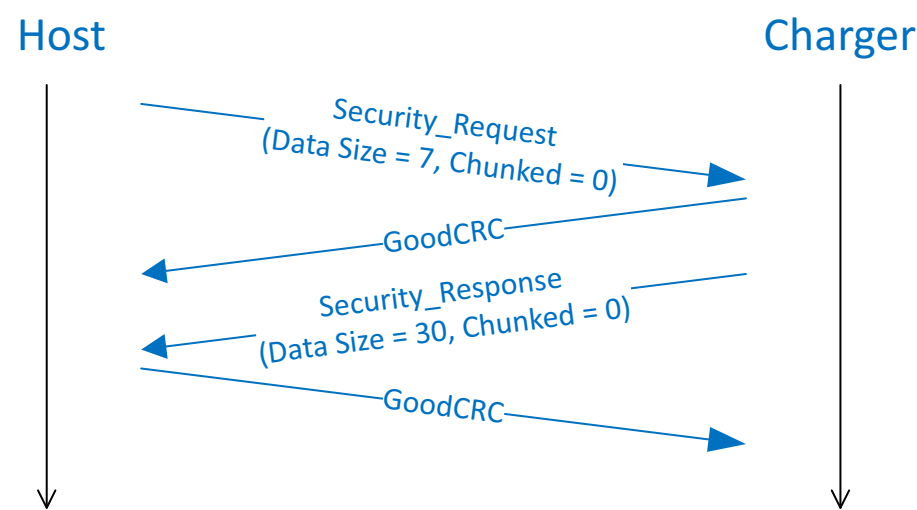


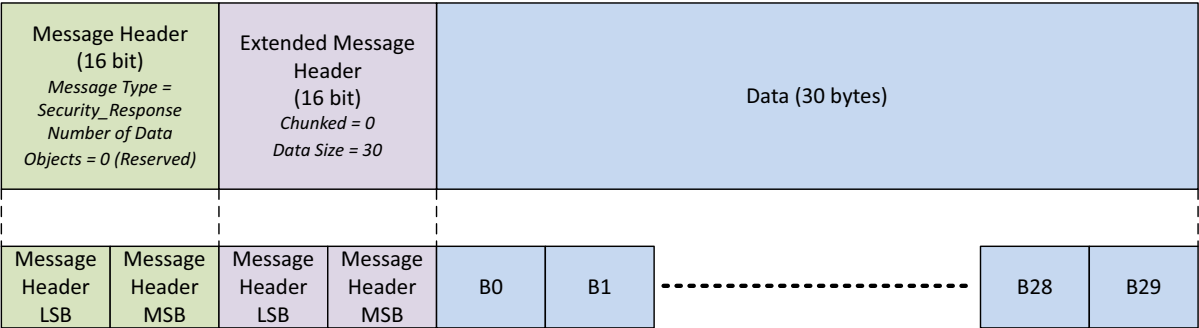
Figure 6.5, "Example byte transmission for Security_Request Message of Data Size 7 (Chunked bit is set to zero)" details the **Security_Request** Message shown in Figure 6.4, "Example Security_Request sequence Unchunked (Chunked bit = 0)". The figure shows the byte ordering on the bus as well as the fact that there is no padding in this case. The **Number of Data Objects** field has a value of 0 since it is **Reserved** when the **Chunked** bit is zero. The **Data Size** field indicates the length of the **Extended Message** when the **Chunked** bit is set to 0, which in this case is 7 bytes.

Figure 6.5 Example byte transmission for Security_Request Message of Data Size 7 (Chunked bit is set to zero)

Message Header (16 bit) Message Type = Security_Request Number of Data Objects = 0 (Reserved)		Extended Message Header (16 bit) Chunked = 0 Data Size = 7		Data (7 bytes)						
Message Header LSB	Message Header MSB	Message Header LSB	Message Header MSB	B0	B1	B2	B3	B4	B5	B6

Figure 6.6, "Example byte transmission for Security_Response Message of Data Size 30 (Chunked bit is set to zero)" details the **Security_Response** Message shown in Figure 6.4, "Example Security_Request sequence Unchunked (Chunked bit = 0)". The figure shows the byte ordering on the bus as well as the fact that there is no padding in this case. The **Number of Data Objects** field has a value of 0 since it is **Reserved** when the **Chunked** bit is zero. The **Data Size** field indicates the length of the **Extended Message** when the **Chunked** bit is set to zero, which in this case is 30 bytes.

Figure 6.6 Example byte transmission for Security_Response Message of Data Size 30 (Chunked bit is set to zero)



6.2.1.2.5.2 Security_Request/Security_Response Chunked Example

Figure 6.7, "Example Security_Request sequence Chunked (Chunked bit = 1)" illustrates a typical sequence for a Security_Request Message responded to by a Security_Response Message using Chunked Extended Messages (Chunked bit is one) between a USB Host and a Charger.

Note: Chunk Number zero in every Extended Message is sent without the need for a Chunk Request, but Chunk Number one and following need to be requested with a Chunk request.

Figure 6.7 Example Security_Request sequence Chunked (Chunked bit = 1)

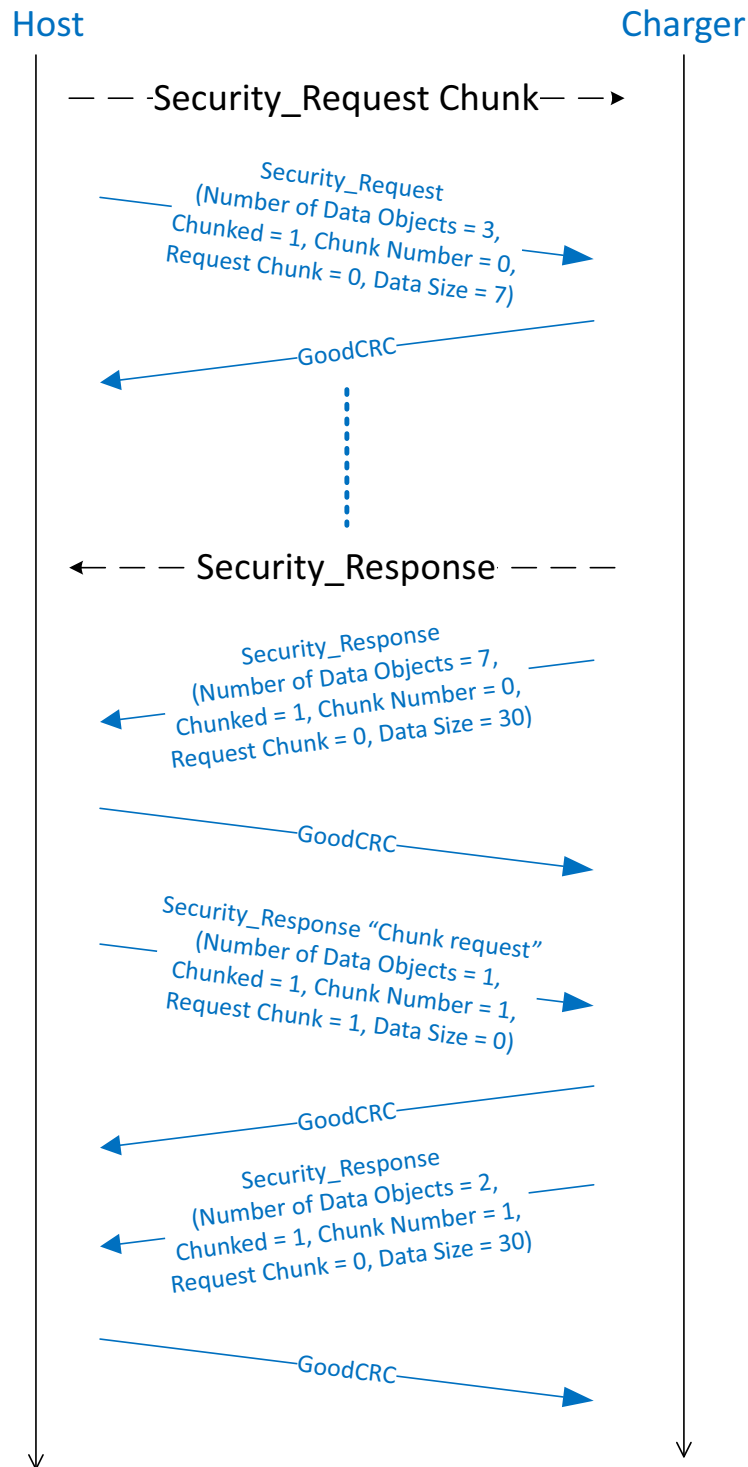


Figure 6.8, "Example Security_Request Message of Data Size 7 (Chunked bit set to 1)" shows the **Security_Request** Message shown in Figure 6.7, "Example Security_Request sequence Chunked (Chunked bit = 1)" in more detail including the byte ordering on the bus and padding. Three bytes of padding have been added to the Message so that the total number of bytes is a multiple of 32-bits, corresponding to 3 Data Objects. The **Number of Data Objects** field is set to 3 to indicate the length of this Chunk. The **Chunk Number** is set to zero and the **Data Size** field is set to 7 to indicate the length of the whole *Extended Message*.

Figure 6.8 Example Security_Request Message of Data Size 7 (Chunked bit set to 1)

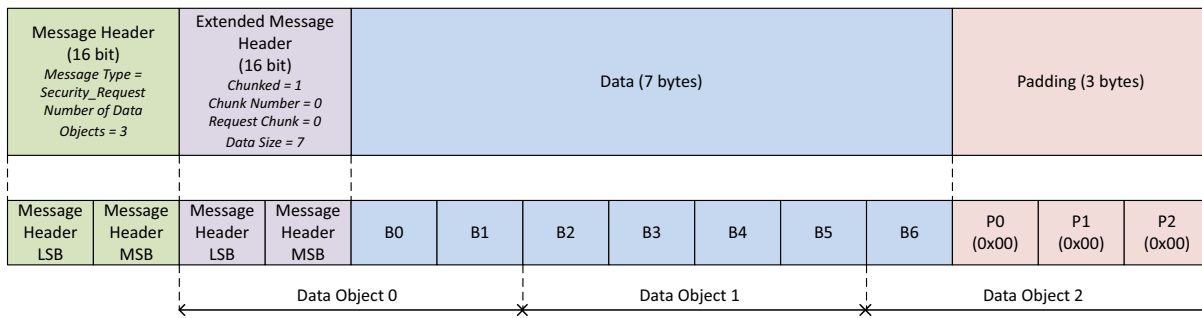


Figure 6.9, "Example Chunk 0 of Security_Response Message of Data Size 30 (Chunked bit set to 1)" shows **Chunk Number** zero of the **Security_Response** Message shown in Figure 6.7, "Example Security_Request sequence Chunked (Chunked bit = 1)" in more detail including the byte ordering on the bus and padding. No padding is need for this **Chunk** since the full 26-byte **Payload** plus 2-byte **Extended Message Header** is a multiple of 32-bits, corresponding to 7 **Data Objects**. The **Number of Data Objects** field is set to 7 to indicate the length of this **Chunk** and the **Data Size** field is set to 30 to indicate the length of the whole **Extended Message**.

Figure 6.9 Example Chunk 0 of Security_Response Message of Data Size 30 (Chunked bit set to 1)

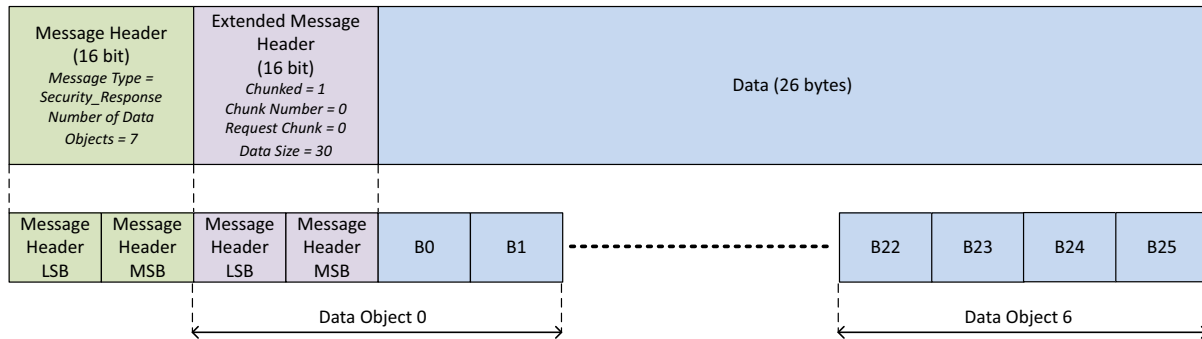


Figure 6.7, "Example Security_Request sequence Chunked (Chunked bit = 1)" shows an example of the **Message** format, byte ordering and padding for the **Security_Response** Message **Chunk** request for **Chunk Number** one shown in Figure 6.7, "Example Security_Request sequence Chunked (Chunked bit = 1)". In the **Chunk** request the **Number of Data Objects** field in the **Message** is set to 1 to indicate that the **Payload** is 32 bits equivalent to 1 data object (see Figure 6.10, "Example byte transmission for a Security_Response Message Chunk request (Chunked bit is set to 1)"). Since the **Chunked** bit is set to 1 the **Chunk** request/Chunk response mechanism is used. The **Message** is a **Chunk** request so the **Request Chunk** bit is set to one, and in this case **Chunk** one is being requested so **Chunk Number** is set to one. **Data Size** is set to zero indicating the length of the **Data Block** being transferred. Two bytes of padding are added to ensure that the **Payload** is a multiple of 32 bits.

Figure 6.10 Example byte transmission for a Security_Response Message Chunk request (Chunked bit is set to 1)

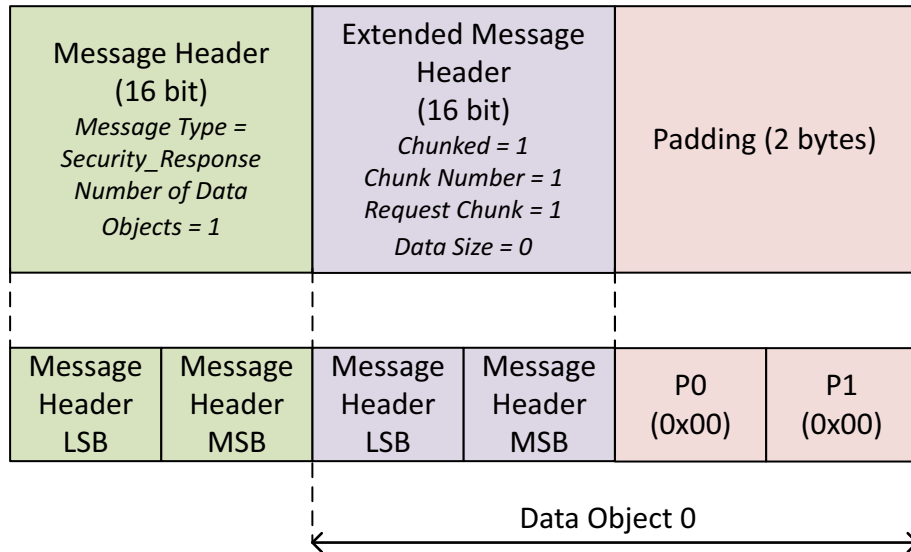
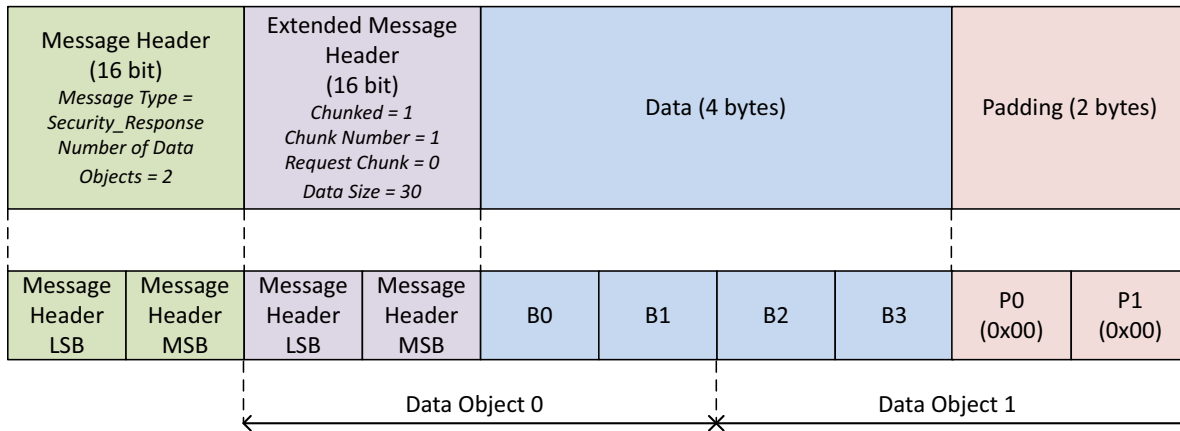


Figure 6.11, "Example Chunk 1 of Security_Response Message of Data Size 30 (Chunked bit set to 1)" shows **Chunk Number** one of the **Security_Response** Message shown in Figure 6.7, "Example Security_Request sequence Chunked (Chunked bit = 1)" in more detail including the byte ordering on the bus and padding. Two bytes of padding are added to ensure that the **Payload** is a multiple of 32 bits, corresponding to 2 **Data Objects**. The **Number of Data Objects** field is set to 2 to indicate the length of this **Chunk** and the **Data Size** field is set to 30 to indicate the length of the whole **Extended Message**.

Figure 6.11 Example Chunk 1 of Security_Response Message of Data Size 30 (Chunked bit set to 1)



6.3 Control Message

A Message is defined as a Control Message when the **Number of Data Objects** field in the Message Header is set to zero. The Control Message consists only of a Message Header and a CRC. The Protocol Layer originates the Control Messages (i.e., **Accept** Message, **Reject** Message etc.).

The Control Message types are specified in the Message Header's **Message Type** field (bits 4...0) and are summarized in [Table 6.5, "Control Message Types"](#). The Sent by column indicates entities which **May** send the given Message (Source, Sink or Cable Plug); entities not listed **Shall Not** issue the corresponding Message. The "Valid Start of Packet" column indicates the Messages which **Shall** only be issued in SOP Packets and the Messages which **May** be issued in SOP* Packets.

Table 6.5 Control Message Types

Bits 4...0	Message Type	Sent by	Description	Valid Start of Packet
0_0000	Reserved	N/A	All values not explicitly defined are Reserved and Shall Not be used.	
0_0001	GoodCRC	Source, Sink or Cable Plug	See Section 6.3.1 .	SOP*
0_0010	GotoMin (Deprecated)	Deprecated	See Section 6.3.2 .	N/A
0_0011	Accept	Source, Sink or Cable Plug	See Section 6.3.3 .	SOP*
0_0100	Reject	Source, Sink or Cable Plug	See Section 6.3.4 .	SOP*
0_0101	Ping (Deprecated)	Deprecated	See Section 6.3.5 .	SOP only
0_0110	PS_RDY	Source or Sink	See Section 6.3.6 .	SOP only
0_0111	Get_Source_Cap	Sink or DRP	See Section 6.3.7 .	SOP only
0_1000	Get_Sink_Cap	Source or DRP	See Section 6.3.8 .	SOP only
0_1001	DR_Swap	Source or Sink	See Section 6.3.9 .	SOP only
0_1010	PR_Swap	Source or Sink	See Section 6.3.10 .	SOP only
0_1011	VCONN_Swap	Source or Sink	See Section 6.3.11 .	SOP only
0_1100	Wait	Source or Sink	See Section 6.3.12 .	SOP only
0_1101	Soft_Reset	Source or Sink	See Section 6.3.13 .	SOP*
0_1110	Data_Reset	Source or Sink	See Section 6.3.14 .	SOP only
0_1111	Data_Reset_Complete	Source or Sink	See Section 6.3.15 .	SOP only
1_0000	Not_Supported	Source, Sink or Cable Plug	See Section 6.3.16 .	SOP*
1_0001	Get_Source_Cap_Extended	Sink or DRP	See Section 6.3.17 .	SOP only
1_0010	Get_Status	Source or Sink	See Section 6.3.18 .	SOP*
1_0011	FR_Swap	Sink ¹	See Section 6.3.19 .	SOP only
1_0100	Get_PPS_Status	Sink	See Section 6.3.20 .	SOP only
1_0101	Get_Country_Codes	Source or Sink	See Section 6.3.21 .	SOP only
1_0110	Get_Sink_Cap_Extended	Source or DRP	See Section 6.3.22 .	SOP only
1_0111	Get_Source_Info	Sink or DRP	See Section 6.3.23 .	SOP Only
1_1000	Get_Revision	Source or Sink	See Section 6.3.24 .	SOP*
1_1001... 1_1111	Reserved	N/A	All values not explicitly defined are Reserved and Shall Not be used.	

1) In this case the Port is providing **vSafe5V** however it will have R_d asserted rather than R_p and sets the **Port Power Role** field to Sink, until the Fast Role Swap AMS has completed.

6.3.1 GoodCRC Message

The **GoodCRC** Message **Shall** be sent by the receiver to acknowledge that the previous *Message* was correctly received (i.e., had a **GoodCRC** Message). The **GoodCRC** Message **Shall** return the *Message*'s **MessageID** so the sender can determine that the correct *Message* is being acknowledged. The first bit of the **GoodCRC** Message **Shall** be returned within **tTransmit** after receipt of the last bit of the previous *Message*.

BIST does not send the **GoodCRC** Message while in a *Continuous BIST Mode* (see [Section 6.4.3, "BIST Message"](#)).

The retry mechanism is triggered when the *Message* sender fails to receive a **GoodCRC** Message before the **CRCReceiveTimer** expires. It is used by the *Message* sender to detect that the *Message* was not correctly received by the *Message* recipient due to noise or other disturbance on the *Configuration Channel (CC)*. The retry mechanism **Shall Not** be used for any other purpose such as a means of gaining time for processing the required response to the received *Message*.

6.3.2 GotoMin Message (Deprecated)

The **GotoMin (Deprecated)** Message has been **Deprecated**. The 0_0010 **Message Type** is no longer **Valid** and **Shall** be responded to by a **Not_Supported** Message.

6.3.3 Accept Message

The **Accept** Message is a **Valid** response in the following cases:

- It **Shall** be sent by the *Source*, in *SPR Mode*, to signal the *Sink* that the *Source* is willing to meet the **Request** Message.
- It **Shall** be sent by the *Source*, in *EPR Mode*, to signal the *Sink* that the *Source* is willing to meet the **EPR_Request** Message.
- It **Shall** be sent by the recipient of the **PR_Swap** Message to signal that it is willing to do a *Power Role Swap* and has begun the *Power Role Swap AMS*.
- It **Shall** be sent by the recipient of the **DR_Swap** Message to signal that it is willing to do a *Data Role Swap* and has begun the *Data Role Swap AMS*.
- It **Shall** be sent by the recipient of the **VCONN_Swap** Message to signal that it is willing to do a *VCONN Swap* and has begun the *VCONN Swap AMS*.
- It **Shall** be sent by the recipient of the **FR_Swap** Message to indicate that it has begun the *Fast Role Swap AMS*.
- It **Shall** be sent by the recipient of the **Soft_Reset** Message to indicate that it has completed its *Soft Reset*.
- It **Shall** be sent by the recipient of the **Enter_USB** Message to indicate that it has begun the *Enter USB AMS*.
- It **Shall** be sent by the recipient of the **Data_Reset** Message to indicate that it has begun the *Data Reset AMS*.

The **Accept** Message **Shall** be sent within **tReceiverResponse** of the receipt of the last bit of the *Message* (see [Section 6.6.2, "SenderResponseTimer"](#)).

6.3.4 Reject Message

The **Reject** Message is a **Valid** response in the following cases:

- It **Shall** be sent to signal the *Sink*, in *SPR Mode*, that the *Source* is unable to meet the **Request** Message. This **May** be due an **Invalid** request or because the *Source* can no longer provide what it previously *Advertised*.

- It **Shall** be sent to signal the *Sink*, in *EPR Mode*, that the *Source* is unable to meet the **EPR_Request Message**. This **May** be due an **Invalid** request or because the *Source* can no longer provide what it previously *Advertised*.
- It **Shall** be sent by the recipient of a **PR_Swap Message** to indicate it is unable to do a *Power Role Swap*.
- It **Shall** be sent by the recipient of a **PR_Swap Message** while in *EPR Mode*.
- It **Shall** be sent by the recipient of a **DR_Swap Message** to indicate it is unable to do a *Data Role Swap*.
- It **Shall** be sent by the recipient of a **VCONN_Swap Message** that is not presently the *VCONN Source*, to indicate it is unable to do a *VCONN Swap*.
- It **Shall** be sent by *UFP* on receiving an **Enter_USB Message** to indicate it is unable to enter the requested *USB Mode*.

The sender of a **Request**, **EPR_Request**, **PR_Swap**, **DR_Swap**, **VCONN_Swap**, or **Enter_USB Message**, on receiving a **Reject Message** response, **Shall Not** send this same *Message* to the recipient until one of the following has occurred:

- A New *Explicit Contract Negotiation* as a result of the *Source* sending a **Source_Capabilities Message** or **EPR_Source_Capabilities Message**. This can be triggered by:
 - The *Source's Device Policy Manager*.
 - A **Get_Source_Cap Message** sent from the *Sink* to the *Source* in *SPR Mode*.
 - An **EPR_Get_Source_Cap Message** sent from the *Sink* to the *Source* in *EPR Mode*.
 - A *Power Role Swap*.
 - A *Soft Reset*.
 - A *Hard Reset*.
 - A *Disconnect/Re-connect*.
- A *Data Role Swap*.
- A *Data Reset*.

The *Sink* **May** send a different **Request Message** to the one which was rejected but **Shall Not** repeat the same **Request Message**, using the same *RDO*, unless there has been a New *Explicit Contract Negotiation*, *Data Role Swap* or *Data Reset* as described above.

The **Reject Message** **Shall** be sent within **tReceiverResponse** of the receipt of the last bit of *Message* (see [Section 6.6.2, "SenderResponseTimer"](#)).

Note: The **Reject Message** is not a **Valid** response when a *Message* is not supported. In this case the **Not_Supported Message** is returned (see [Section 6.3.16, "Not Supported Message"](#)).

6.3.5 Ping Message

The **Ping (Deprecated) Message** has been deprecated. The 0_0101 **Message Type** is no longer **Valid**.

A *Port* that receives a **Ping (Deprecated) Message** **May** respond with a **Not_Supported Message** or **Ignore** the **Ping (Deprecated) Message**. A *Cable Plug* that receives a **Ping (Deprecated) Message** **Shall Ignore** the **Ping (Deprecated) Message**.

6.3.6 PS_RDY Message

The **PS_RDY Message** **Shall** be sent by the *Source* (or by both the *New Sink* and *New Source* during the *Power Role Swap AMS* or *Fast Role Swap AMS*) to indicate its power supply has reached the desired operating condition (see [Section 8.3.2.2, "Power Negotiation"](#)).

6.3.7 Get_Source_Cap Message

The **Get_Source_Cap** (Get Source Capabilities) Message **May** be sent by a Port to request the Source Capabilities and Dual-Role Power capability of its Port Partner (e.g., Dual-Role Power capable). The Port **Shall** respond by returning a **Source_Capabilities** Message (see [Section 6.4.1.5, "SPR Source Capabilities Message"](#)).

6.3.8 Get_Sink_Cap Message

The **Get_Sink_Cap** (Get Sink Capabilities) Message **May** be sent by a Port to request the Sink Capabilities and Dual-Role Power capability of its Port Partner (e.g., Dual-Role Power capable). The Port **Shall** respond by returning a **Sink_Capabilities** Message (see [Section 6.4.1.6, "SPR Sink Capabilities Message"](#)).

6.3.9 DR_Swap Message

The **DR_Swap** Message is used to exchange DFP and UFP operation between Port Partners while maintaining the direction of power flow over VBUS. The Data Role Swap process can be used by Port Partners whether or not they support USB Communications capability. A DFP that supports USB Communication capability starts as the USB Host on Attachment. A UFP that supports USB Communication capability starts as the USB Device on Attachment.

[USB Type-C 2.4] Dual-Role Data (DRD) Ports **Shall** have the capability to perform a Data Role Swap from the **PE_SRC_Ready** or **PE_SNK_Ready** states. DFPs and UFPs **May** have the capability to perform a Data Role Swap from the **PE_SRC_Ready** or **PE_SNK_Ready** states. A Data Role Swap **Shall** be regarded in the same way as a cable Detach/ Re-attach in relation to any USB Communication which is ongoing between the Port Partners. If there are any Active Modes between the Port Partners when a **DR_Swap** Message is received, then a Hard Reset **Shall** be performed (see [Section 6.4.4.3.4, "Enter Mode Command"](#)). If the Cable Plug has any Active Modes then the DFP **Shall Not** issue a **DR_Swap** Message and **Shall** cause all Active Modes in the Cable Plug to be exited before accepting a Data Role Swap request.

The source of VBUS and VCONN Source **Shall** remain unchanged as well as the R_p/R_d resistors on the CC wire during the Data Role Swap process.

The **DR_Swap** Message **May** be sent by either Port Partner. The recipient of the **DR_Swap** Message **Shall** respond by sending an **Accept** Message, a **Wait** Message or a **Reject** Message (see [Section 6.9, "Accept, Reject and Wait"](#)).

- If an **Accept** Message is sent, the Source and Sink **Shall** exchange Data Roles.
- If a **Reject** Message is sent, the requester is informed that the recipient is unable, or unwilling, to do a Data Role Swap and no action **Shall** be taken.
- If a **Wait** Message is sent, the requester is informed that a Data Role Swap might be possible in the future but that no immediate action **Shall** be taken.

Before a Data Role Swap the initial DFP **Shall** have its **Port Data Role** bit set to DFP, and the initial UFP **Shall** have its **Port Data Role** bit set to UFP.

After a successful Data Role Swap the DFP/Host **Shall** become the UFP/Device and vice-versa; the new DFP **Shall** have its **Port Data Role** bit set to DFP, and the new UFP **Shall** have its **Port Data Role** bit set to UFP. Where USB Communication is supported by both Port Partners a USB data connection **Should** be established according to the new Data Roles.

If the Data Role Swap, after having been accepted by the Port Partner, is subsequently not successful, in order to attempt a re-establishment of the connection, USB Type-C Error Recovery actions, such as disconnect, as defined in **[USB Type-C 2.4]** will be necessary.

See [Section 8.3.2.9, "Data Role Swap"](#).

6.3.10 PR_Swap Message

The **PR_Swap** Message **May** be sent by either Port Partner to request an exchange of Power Roles. The recipient of the Message **Shall** respond by sending an **Accept** Message, a **Wait** Message or a **Reject** Message (see [Section 6.9, "Accept, Reject and Wait"](#)).

- If an **Accept** Message is sent, the Source and Sink **Shall** do a Power Role Swap.
- If a **Reject** Message is sent, the requester is informed that the recipient is unable, or unwilling, to do a Power Role Swap and no action **Shall** be taken.
- If a **Wait** Message is sent, the requester is informed that a Power Role Swap might be possible in the future but that no immediate action **Shall** be taken.

The **PR_Swap** Message **Shall Not** be sent while in *EPR Mode*. While in *EPR Mode* if a Power Role Swap is required, an *EPR Mode* exit **Shall** be done first.

After a successful Power Role Swap the Port Partners **Shall** reset their respective Protocol Layers (equivalent to a Soft Reset): resetting their **MessageIDCounter**, **RetryCounter** and Protocol Layer state machines before attempting to establish the First Explicit Contract. At this point the New Source **Shall** also reset its **CapsCounter**.

The New Source **Shall** have R_p asserted on the CC wire and the New Sink **Shall** have R_d asserted on the CC wire as defined in [\[USB Type-C 2.4\]](#). When performing a Power Role Swap from Source to Sink, the Port **Shall** change its CC wire resistor from R_p to R_d . When performing a Power Role Swap from Sink to Source, the Port **Shall** change its CC wire resistor from R_d to R_p . The DFP (Host), UFP (Device) Data Roles and VCONN Source **Shall** remain unchanged by the Power Role Swap process.

Note: During the Power Role Swap process the Initial Sink does not disconnect even though *VBUS* drops below *vSafe5V*.

For more information regarding the Power Role Swap, refer to:

- [Section 7.3.2, "Transitions Caused by Power Role Swap"](#)
- [Section 8.3.2.5, "Data Reset"](#).
- [Section 8.3.3.19.3, "Policy Engine in Source to Sink Power Role Swap State Diagram"](#).
- [Section 8.3.3.19.4, "Policy Engine in Sink to Source Power Role Swap State Diagram"](#).
- [Section 9.1.2, "Mapping to USB Device States"](#).

6.3.11 VCONN_Swap Message

The **VCONN_Swap** Message **Shall** be supported by any Port that can operate as a VCONN Source.

The **VCONN_Swap** Message **May** be sent by either Port Partner to request an exchange of VCONN Source. The recipient of the Message **Shall** respond by sending an **Accept** Message, **Reject** Message, **Wait** Message (see [Section 6.9, "Accept, Reject and Wait"](#)) or **Not_Supported** Message.

- If an **Accept** Message is sent, the Port Partners **Shall** perform a VCONN Swap. The new VCONN Source **Shall** send a **PS_RDY** Message within *tVCONNSourceOn* to indicate that it is now sourcing VCONN. The initial VCONN Source **Shall** cease sourcing VCONN within *tVCONNSourceOff* of receipt of the last bit of the **EOP** of the **PS_RDY** Message.
- If a **Reject** Message is sent, the requester is informed that the recipient is unable, or unwilling, to do a VCONN Swap and no action **Shall** be taken. A **Reject** Message **Shall** only be sent by the Port that is not presently the VCONN Source in response to a **VCONN_Swap** Message. The Port that is presently the VCONN Source **Shall Not** send a **Reject** Message in response to **VCONN_Swap** Message.
- If a **Wait** Message is sent, the requester is informed that a VCONN Swap might be possible in the future but that no immediate action **Shall** be taken. A Port after losing the VCONN Source role due to incoming VCONN Swap request **Shall Not** initiate a VCONN Swap until at least *tVCONNSwapDelayDFP*/*tVCONNSwapDelayUFP* after completing the previous VCONN Swap AMS.
- If a **Not_Supported** Message is sent, the requester is informed that VCONN Swap is not supported. The Port that is not presently the VCONN Source **May** turn on VCONN when a **Not_Supported** Message is received in response to a **VCONN_Swap** Message.

The *DFP (Host)*, *UFP (Device)* *Data Roles* and *Source* of *VBUS* **shall** remain unchanged as well as the R_p/R_d resistors on the *CC* wire during the *VCONN Swap* process.

VCONN **shall** be continually sourced during the *VCONN Swap* process to maintain power to the *Cable Plug(s)* i.e., make before break.

Before communicating with a *Cable Plug* a *Port* **shall** ensure that it is the *VCONN Source* and that the *Cable Plugs* are powered, by performing a *VCONN Swap* if necessary. Since it cannot be guaranteed that the present *VCONN Source* is supplying *VCONN*, the only means to ensure that the *Cable Plugs* are powered is for a *Port* wishing to communicate with a *Cable Plug* to become the *VCONN Source*. If a **Not Supported** Message is returned in response to the *VCONN_Swap* Message, then the *Port* is allowed to become the *VCONN Source* until a *Hard Reset* or *Detach*.

A *VCONN Source* that is also a *Source* can attempt to send a **Discover Identity** Command using *SOP'* to a *Cable Plug* prior to the establishment of the *First Explicit Contract*.

Note: Even when it is presently the *VCONN Source*, the *Sink* is not permitted to initiate an *AMS* with a *Cable Plug* unless R_p is set to **SinkTxOK** (see [Section 6.9, "Accept, Reject and Wait"](#)).

6.3.12 Wait Message

The **Wait** Message is a **Valid** response to one of the following Messages:

- It **shall** be sent to signal the *Sink*, in response to a **Request** Message in *SPR Mode* during *Negotiation*, to indicate that the *Source* is currently unable to meet the request.
- It **shall** be sent to signal the *Sink*, in response to a **EPR_Request** Message in *EPR Mode* during *Negotiation*, to indicate that the *Source* is currently unable to meet the request.
- It **shall** be sent by the recipient of a **PR_Swap** Message to indicate it is currently unable to do a *Power Role Swap*.
- It **shall** be sent by the recipient of a **DR_Swap** Message to indicate it is currently unable to do a *Data Role Swap*.
- It **shall** be sent by the recipient of a **VCONN_Swap** Message that is not presently the *VCONN Source* to indicate it is currently unable to do a *VCONN Swap*.
- It **shall** be sent by the recipient of an **Enter_USB** Message to indicate it is currently unable to enter the requested *USB Mode*.

The **Wait** Message **shall** be sent within **tReceiverResponse** of the receipt of the last bit of the Message (see [Section 6.9, "Accept, Reject and Wait"](#)).

6.3.12.1 Wait in response to a Request Message

The **Wait** Message allows the *Source* time to recover the power it requires to meet the request, e.g., through *Re-negotiation* with other *Sinks* or an upstream *Source*. A *Source* **should** only send a **Wait** Message in response to a *Request* Message when an *Explicit Contract* exists between the *Port Partners*.

The *Sink* is allowed to repeat the *Request* Message using the **SinkRequestTimer** and **shall** ensure that there is **tSinkRequest** after receiving the **Wait** Message before sending another *Request* Message.

6.3.12.2 Wait in response to a PR_Swap Message

The **Wait** Message is used when responding to a **PR_Swap** Message to indicate that a *Power Role Swap* might be possible in the future. This can occur in any case where the device receiving the **PR_Swap** Message needs to evaluate the request further e.g., by requesting *Sink Capabilities* from the originator of the **PR_Swap** Message. Once it has completed this evaluation one of the *Port Partners* **should** initiate the *Power Role Swap* process again by sending a **PR_Swap** Message.

The **Wait** Message is also used where a *Hub* is operating in hybrid mode when a request cannot be satisfied (see [\[UCSI\]](#)).

A Port that receives a **Wait Message** in response to a **PR_Swap Message** **Shall** wait **tPRSwapWait** after receiving the **Wait Message** before sending another **PR_Swap Message**.

6.3.12.3 Wait in response to a DR_Swap Message

The **Wait Message** is used when responding to a **DR_Swap Message** to indicate that a **Data Role Swap** might be possible in the future. This can occur in any case where the device receiving the **DR_Swap Message** needs to evaluate the request further. Once it has completed this evaluation one of the **Port Partners** **Should** initiate the **Data Role Swap** process again by sending a **DR_Swap Message**.

A Port that receives a **Wait Message** in response to a **DR_Swap Message** **Shall** wait **tDRSwapWait** after receiving the **Wait Message** before sending another **DR_Swap Message**.

6.3.12.4 Wait in response to a VCONN_Swap Message

The **Wait Message** is used when responding to a **VCONN_Swap Message** to indicate that a **VCONN_Swap** might be possible in the future. This can occur in any case where the device receiving the **VCONN_Swap Message** needs to evaluate the request further. Once it has completed this evaluation one of the **Port Partners** **Should** initiate the **VCONN Swap** process again by sending a **VCONN_Swap Message**.

A Port that receives a **Wait Message** in response to a **VCONN_Swap Message** **Shall** wait **tVCONNSwapWait** after receiving the **Wait Message** before sending another **VCONN_Swap Message**.

A Port that is currently the **VCONN Source** **Shall** respond with an **Accept Message** (rather than a **Wait Message**) if the **Port Partner's Revision** and **Version**, as reported in the **Revision Message**, is earlier than R3.2 V1.1. A **Port Partner** supporting an earlier **Revision** and **Version** will not expect a **Wait Message** and will generate a **Soft Reset** in response.

6.3.12.5 Wait in response to an Enter_USB Message

The **Wait Message** is used, by the **UFP**, when responding to an **Enter_USB Message** to indicate that entering the requested **USB Mode** might be possible in the future. This can occur, for example, in any case where the **UFP** needs to **Negotiate** more power to enter the mode. Once the **UFP** has completed this the **DFP** **Should** initiate the **Enter USB** process again by sending an **Enter_USB Message**.

A **DFP** that receives a **Wait Message** in response to an **Enter_USB Message** **Shall** wait **tEnterUSBWait** after receiving the **Wait Message** before sending another **Enter_USB Message**.

6.3.13 Soft Reset Message

A **Soft_Reset Message** **May** be initiated by either the **Source** or **Sink** to its **Port Partner** requesting a **Soft Reset**. The **Soft_Reset Message** **Shall** cause a **Soft Reset** of the connected **Port Pair** (see [Section 6.8.1, "Soft Reset and Protocol Error"](#)). If the **Soft_Reset Message** fails a **Hard Reset** **Shall** be initiated within **tHardReset** of the last **CRCReceiveTimer** expiring after **nRetryCount** retries have been completed.

A **Soft_Reset Message** is used to recover from **Protocol Layer** errors; putting the **Message** counters to a known state to regain **Message** synchronization. The **Soft_Reset Message** has no effect on the **Source** or **Sink**; that is the previously **Negotiated** direction. Voltage and current remain unchanged. **Modal Operation** is unaffected by **Soft Reset**. However after a **Soft Reset** has completed, an **Explicit Contract Negotiation** occurs, in order to re-establish **PD Communication** and to bring state operation for both **Port Partners** back to either the **PE_SNK_Ready** or **PE_SRC_Ready** states as appropriate (see [Section 8.3.3.4, "SOP Soft Reset and Protocol Error State Diagrams"](#)).

A **Soft_Reset Message** **May** be sent by either the **Source** or **Sink** when there is a **Message** synchronization error. If the error is not corrected by the **Soft Reset**, **Hard Reset Signaling** **Shall** be issued (see [Section 6.8.3, "Hard Reset"](#)).

A **Soft_Reset Message** **Shall** be targeted at a specific entity depending on the type of **SOP* Packet** used. **Soft_Reset Messages** sent using **SOP Packets** **Shall** **Soft Reset** the **Port Partner** only. **Soft_Reset Messages** sent using **SOP' Packet/ SOP" Packets** **Shall** **Soft Reset** the corresponding **Cable Plug** only.

After a **VCONN Swap** the **VCONN Source** needs to reset the **Cable Plug's Protocol Layer** to ensure **MessageID** synchronization. If after a **VCONN Swap** the **VCONN Source** wants to communicate with a **Cable Plug** using **SOP' Packets**, it **Shall** issue a **Soft_Reset Message** using a **SOP' Packet** in order to reset the **Cable Plug's Protocol Layer**. If

the *VCONN Source* wants to communicate with a *Cable Plug* using *SOP* Packets, it **Shall** issue a **Soft_Reset Message** using a *SOP* Packet in order to reset the *Cable Plug's Protocol Layer*.

6.3.14 Data_Reset Message

The **Data_Reset Message** **May** be sent by either the *DFP* or *UFP* and **Shall** reset the USB data connection and exit all *Alternate Modes* with its *Port Partner* while preserving the power on *VBUS*. *USB4[®] Mode* capable ports **Shall** support the **Data_Reset Message** and other ports **May** support the **Data_Reset Message**.

The **Data_Reset Message** **Shall Not** change the existing:

- Power Contract
- Data Roles (i.e., which *Port* is the *DFP* or *UFP*)

The receiver of the **Data_Reset Message** **Shall** respond by sending an **Accept Message** and then follow the process outlined in the following steps. Neither the sender nor receiver **Shall** initiate a *VCONN Swap* until the *Data Reset* process is complete, and the **Data_Reset_Complete Message** has been sent. Following receipt of the **Accept Message**, or **GoodCRC** following the **Accept**, depending which *Port* sends the **Data_Reset Message**:

- 1) The *DFP* **Shall**:
 - Disconnect the *Port's* [USB 2.0] D+/D- signals.
 - If operating in [USB 3.2] remove the *Port's* Rx Terminations (see [USB 3.2]).
 - If operating in [USB4] drive the *Port's* SBTX to a logic low (see [USB4]).
- 2) Both the *DFP* and *UFP* **Shall** exit all *Alternate Modes* if any.
- 3) Reset the cable:
 - If the *VCONN Source Port* is also the *UFP*, then it **Shall** run the *UFP VCONN Power Cycle* process described in [Section 7.1.15.1, "UFP VCONN Power Cycle"](#).
 - If the *VCONN Source Port* is also the *DFP*, then it **Shall** run the *DFP VCONN Power Cycle* process described in [Section 7.1.15.2, "DFP VCONN Power Cycle"](#).
 - The *DFP* **Shall** exit the *VCONN Power Cycle* process as the *VCONN Source* and be sourcing *VCONN*.
- 4) After **tDataReset** the *DFP* **Shall**:
 - Reconnect the [USB 2.0] D+/D- signals.
 - If the *Port* was operating in [USB 3.2] or [USB4] reapply the *Port's* Rx Terminations (see [USB 3.2]).
- 5) The *Data Reset* process is complete; the *DFP* **Shall** send a **Data_Reset_Complete Message** and enter the *USB4[®] Discovery and Entry Flow* (See [USB Type-C 2.4]).

If the *Initiator* of the **Data_Reset Message** does not receive a **Valid** response within **tSenderResponse** it **Shall** enter the **ErrorRecovery State**.

6.3.15 Data_Reset_Complete Message

The **Data_Reset_Complete Message** **Shall** be sent by the *DFP* to the *UFP* to indicate the completion of the *Data Reset* process (see [Section 6.3.14, "Data_Reset Message"](#)).

6.3.16 Not_Supported Message

The **Not_Supported Message** **Shall** be sent by a *Port* or *Cable Plug* in response to any *Message* it does not support. Returning a **Not_Supported Message** is assumed in this specification and has not been called out explicitly except in [Section 6.13, "Message Applicability"](#) which defines cases where the **Not_Supported Message** is returned.

6.3.17 Get_Source_Cap_Extended Message

The **Get_Source_Cap_Extended** Message is sent by a *Port* to request additional information about a *Port's Source Capabilities*. The *Port* **Shall** respond by returning a **Source_Capabilities_Extended** Message (see [Section 6.5.1, "Source Capabilities Extended Message"](#)).

6.3.18 Get_Status Message

The **Get_Status** Message is sent by a *Port* using *SOP* to request the *Port Partner's* present status.

The *Port Partner* **Shall** respond by returning a **Status** Message (see [Section 6.5.2, "Status Message"](#)). A *Port* that receives an **Alert** Message (see [Section 6.4.6, "Alert Message"](#)) indicates that the *Source* or *Sink's* Status has changed and **Should** be re-read using a **Get_Status** Message.

The **Get_Status** Message **May** also be sent to an *Active Cable* to get its present status using *SOP'/SOP''*.

The *Active Cable* **Shall** respond by returning a **Status** Message (see [Section 6.5.2, "Status Message"](#)).

6.3.19 FR_Swap Message

The **FR_Swap** Message **Shall** be sent by the *New Source* within **tFRSwapInit** after it has detected a *Fast Role Swap* signal (see [Section 5.8.6.3, "Fast Role Swap Detection"](#) and [Section 6.6.17.3, "tFRSwapInit"](#)). The *Fast Role Swap AMS* is necessary to apply R_p to the *New Source* and R_d to the *New Sink* and to re-synchronize the state machines. The **tFRSwapInit** time **Shall** be measured from the time the *Fast Role Swap Request* has been sent for **tFRSwapRx** (max) until the last bit of the **EOP** of the **FR_Swap** Message has been transmitted by the *PHY Layer*.

The recipient of the **FR_Swap** Message **Shall** respond by sending an **Accept** Message.

After a successful *Fast Role Swap* the *Port Partners* **Shall** reset their respective *Protocol Layers* (equivalent to a *Soft Reset*): resetting their **MessageIDCounter**, **RetryCounter** and *Protocol Layer* state machines before attempting to establish the *First Explicit Contract*. At this point the *Source* **Shall** also reset its **CapsCounter**.

This ensures that only the *Cable Plug* responds with a **GoodCRC** Message to the **Discover Identity** Command.

Prior to the *Fast Role Swap AMS*, the *New Source* **Shall** have R_d asserted on the *CC* wire and the *New Sink* **Shall** have R_p asserted on the *CC* wire.

Note: This is an incorrect assignment of R_p/R_d (since R_p follows the *Source* and R_d follows the *Sink* as defined in [\[USB Type-C 2.4\]](#)) that is corrected by the *Fast Role Swap AMS*.

During the *Fast Role Swap AMS*, the *New Source* **Shall** change its *CC* wire resistor from R_d to R_p and the *New Sink* **Shall** change its *CC* wire resistor from R_p to R_d . The *DFP (Host)*, *UFP (Device)* *Data Roles* and *VCONN Source* **Shall** remain unchanged during the *Fast Role Swap* process.

The *Initial Source* **Should** avoid being the *VCONN Source* (by using the *VCONN Swap* process) whenever not actively communicating with the cable, since it is difficult for the *Initial Source* to maintain *VCONN* power during the *Fast Role Swap* process.

Note: A *Fast Role Swap* is a "best effort" solution to a situation where a *PDUSB Device* has lost its external power. This process can occur at any time, even during an *AMS* in which case error handling such as *Hard Reset* or [\[USB Type-C 2.4\]](#) *Error Recovery* will be triggered.

Note: During the *Fast Role Swap* process the *Initial Sink* does not disconnect even though *VBUS* drops below **vSafe5V**.

For more information regarding the *Fast Role Swap* process, refer to:

- [Section 7.1.13, "Fast Role Swap"](#)
- [Section 7.2.10, "Fast Role Swap"](#)
- [Section 8.3.3.19.5, "Policy Engine in Source to Sink Fast Role Swap State Diagram"](#)
- [Section 8.3.3.19.6, "Policy Engine in Sink to Source Fast Role Swap State Diagram"](#)

- [Section 9.1.2, "Mapping to USB Device States"](#) for *V_{BUS}* mapping to USB states.

6.3.20 Get_PPS_Status

The **Get_PPS_Status** Message is sent by the *Sink* to request additional information about a *Source*'s status. The *Port* **Shall** respond by returning a **PPS_Status** Message (see [Section 6.5.10, "PPS_Status Message"](#)).

6.3.21 Get_Country_Codes

The **Get_Country_Codes** Message is sent by a *Port* to request the alpha-2 country codes its *Port Partner* supports as defined in [\[ISO 3166\]](#). The *Port Partner* **Shall** respond by returning a **Country_Codes** Message (see [Section 6.5.11, "Country_Codes Message"](#)).

6.3.22 Get_Sink_Cap_Extended Message

The **Get_Sink_Cap_Extended** (Get Sink Capabilities Extended) Message is sent by a *Port* to request additional information about a *Port*'s *Sink Capabilities*. The *Port* **Shall** respond by returning a **Sink_Capabilities_Extended** Message (see [Section 6.5.13, "Sink_Capabilities_Extended Message"](#)).

6.3.23 Get_Source_Info Message

The **Get_Source_Info** Message is sent by a *Port* to request the type, maximum *Capabilities* and present *Capabilities* of the *Port* when it is operating as a *Source*. The *Port* **Shall** respond by returning the **Source_Info** Message (See [Section 6.4.11, "Source_Info Message"](#)).

6.3.24 Get_Revision Message

The **Get_Revision** Message is sent by a *Port* using *SOP* to request the *Revision* and *Version* of the Power Delivery Specification its *Port Partner* supports.

The *Port Partner* **Shall** respond by returning a *Revision Message* (See [Section 6.4.12, "Revision Message"](#)).

The **Get_Revision** Message **May** also be sent to a *Cable Plug* to request the *Revision* and *Version* of the Power Delivery Specification it supports using *SOP'*/*SOP*".

The *Active Cable* **Shall** respond by returning a *Revision Message* (see [Section 6.4.12, "Revision Message"](#)).

6.4 Data Message

A *Data Message* **Shall** consist of a *Message Header* and be followed by one or more *Data Objects*. *Data Messages* are easily identifiable because the **Number of Data Objects** field in the *Message Header* is a non-zero value.

There are many types of *Data Objects* used to compose *Data Messages*. Some examples are:

- *Power Data Object (PDO)* used to expose a *Source Port's* power *Capabilities* or a *Sink's* power requirements.
- *Request Data Object (RDO)* used by a *Sink Port* to *Negotiate* an *Explicit Contract*.
- *Vendor Data Object (VDO)* used to convey vendor specific information.
- *BIST Data Object (BDO)* used for *PHY Layer* compliance testing.
- *Battery Status Data Object (BSDO)* used to convey *Battery* status information.
- *Alert Data Object (ADO)* used to indicate events occurring on the *Source* or *Sink*.

The type of *Data Object* being used in a *Data Message* is defined by the *Message Header's* **Message Type** field and is summarized in [Table 6.6, "Data Message Types"](#). The Sent by column indicates entities which **May** send the given *Message* (*Source*, *Sink* or *Cable Plug*); entities not listed **Shall Not** issue the corresponding *Message*. The "Valid Start of Packet" column indicates the *Messages* which **Shall** only be issued in *SOP Packets* and the *Messages* which **May** be issued in *SOP* Packets*.

Table 6.6 Data Message Types

Bits 4...0	Type	Sent by	Description	Valid Start of Packet
0_0000	Reserved	N/A	All values not explicitly defined are Reserved and Shall Not be used.	N/A
0_0001	Source_Capabilities	Source or Dual-Role Power	See Section 6.4.1.5	SOP only
0_0010	Request	Sink only	See Section 6.4.2	SOP only
0_0011	BIST	Tester, Source or Sink	See Section 6.4.3	SOP*
0_0100	Sink_Capabilities	Sink or Dual-Role Power	See Section 6.4.2	SOP only
0_0101	Battery_Status	Source or Sink	See Section 6.4.5	SOP only
0_0110	Alert	Source or Sink	See Section 6.4.6	SOP only
0_0111	Get_Country_Info	Source or Sink	See Section 6.4.7	SOP only
0_1000	Enter_USB	DFP	See Section 6.4.8	SOP*
0_1001	EPR_Request	Sink	See Section 6.4.9	SOP only
0_1010	EPR_Mode	Source or Sink	See Section 6.4.10	SOP only
0_1011	Source_Info	Source	See Section 6.4.11	SOP only
0_1100	Revision	Source, Sink or Cable Plug	See Section 6.4.12	SOP*
0_1101...0_1110	Reserved	N/A	All values not explicitly defined are Reserved and Shall Not be used.	N/A
0_1111	Vendor_Defined	Source, Sink or Cable Plug	See Section 6.4.4	SOP*
1_0000...1_1111	Reserved	N/A	All values not explicitly defined are Reserved and Shall Not be used.	N/A

6.4.1 Capabilities Message

There are two distinct *Capabilities Messages*: one used while in *SPR Mode* and another while in *EPR Mode*. This section defines the *Capabilities Messages* specific to the *SPR Mode* and [Section 6.5.15, "EPR Capabilities Message"](#) defines the *Capabilities Messages* specific to the *EPR Mode*.

6.4.1.1 Power Data Objects

Sections [Section 6.4.1.5, "SPR Source Capabilities Message"](#) and [Section 7.1.3, "Types of Sources"](#) describes the *Power Data Objects* (PDOs) used in the construction of a *Capabilities Message* for both *SPR Mode* and *EPR Mode*.

There are three types of *Power Data Objects*. They contain additional information beyond that encoded in the *Message Header* to identify each of the three types of *Power Data Objects*:

- *Fixed Supply* is used to expose well-regulated fixed voltage power supplies.
- *Variable Supply* is used to expose very poorly regulated power supplies.
- *Battery Supply* is used to expose batteries that can be directly connected to *VBUS*.

There are three types of *Augmented Power Data Objects*:

- *SPR PPS* is used to expose a power supply whose output voltage can be programmatically adjusted over the *Advertised* voltage range and limited by the *Source* to a programmable current limit.
- *SPR AVS* and *EPR AVS* are used to expose a power supply whose output voltage can be adjusted over the *Advertised* voltage range but otherwise is equivalent to a *Fixed Supply* (*AVS* does not support a programmable current limit).

Power Data Objects are also used to expose additional *Capabilities* that **May** be utilized, such as in the case of a *Power Role Swap*.

A list of one or more *Power Data Objects* **Shall** be sent by the *Source* to convey its *Capabilities*. The *Sink* **May** then request one of these *Capabilities* by returning a *Request Data Object* that contains an index to a *Power Data Object*, to *Negotiate* a mutually agreeable *Explicit Contract*.

Where Maximum and Minimum voltage and current values are given in *PDOs* these **Shall** be taken to be absolute values.

The *Source* and *Sink* **Shall Not Negotiate** a power level that would allow the current to exceed the maximum current supported by their receptacles or the *Attached* plug (see [\[USB Type-C 2.4\]](#)). The *Source* **Shall** limit its offered *Capabilities* to the maximum current supported by its receptacle and *Attached* plug. A *Sink* **Shall** only make a request from any of the *Capabilities* offered by the *Source*. For further details see [Section 4.4, "Cable Type Detection"](#).

Sources expose their power *Capabilities* by sending a *Source_Capabilities* Message. *Sinks* expose their power requirements by sending a *Sink_Capabilities* Message. Both are composed of several 32-bit *Power Data Objects* (see [Table 6.7, "Power Data Object"](#)).

Table 6.7 Power Data Object

Bit(s)	Description	
	Value	Parameter
B31...30	00b	<i>Fixed Supply</i> ($V_{min} = V_{max}$)
	01b	<i>Battery</i>
	10b	<i>Variable Supply</i> (non-Battery)
	11b	<i>Augmented Power Data Object</i> (APDO)
B29...0	Specific <i>Power Capabilities</i> are described by the <i>PDOs</i> in the following sections.	

The *Augmented Power Data Object (APDO)* is defined to allow support for more than the four *PDO* types by extending the *Power Data Object* field from 2 to 4 bits when the B31...B30 are 11b. The generic *APDO* structure is shown in [Table 6.8, "Augmented Power Data Object"](#).

Table 6.8 Augmented Power Data Object

Bit(s)	Description	
	Value	Parameter
B31...30	11b	<i>Augmented Power Data Object (APDO)</i>
B29...28	00b	<i>SPR PPS</i>
	01b	<i>EPR AVS</i>
	10b	<i>SPR AVS</i>
	11b	<i>Reserved</i>
B27...0	Specific <i>Power Capabilities</i> are described by the <i>APDOs</i> in the following sections.	

6.4.1.2 Source Power Data Objects

This section lists the types of *PDOs* a *Source* can use in an *SPR Capabilities* or *EPR Capabilities Message*.

6.4.1.2.1 Fixed Supply Power Data Object

[Table 6.9, "Fixed Supply PDO – Source"](#) describes the *Fixed Supply* (00b) *PDO*. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

Since all *USB Providers* support **vSafe5V**, the required **vSafe5V Fixed Supply Power Data Object** is also used to convey additional information that is returned in bits 29...23. All other *Fixed Supply Power Data Objects* **Shall** set bits 29...23 to zero.

For a *Source* offering no *Capabilities*, the **Voltage** field (B19...10) **Shall** be set to 5V and the **Maximum Current** field **Shall** be set to 0mA. This is used in cases such as a *Dual-Role Power* device which offers no *Capabilities* in its default *Power Role* or when external power is required to offer power.

When a *Source* wants a *Sink*, consuming power from *VBUS*, to go to its lowest power state, the **Voltage** field (B19...10) **Shall** be set to 5V and the **Maximum Current** field **Shall** be set to 0mA. This is used in cases where the *Source* wants the *Sink* to draw **pSnkSusp**.

Table 6.9 Fixed Supply PDO – Source

Bit(s)	Field	Description
B31...30	<i>Fixed Supply</i>	00b - <i>Fixed Supply PDO</i>
B29	Dual-Role Power	Set to '1' for <i>Dual-Role Power</i> device.
B28	USB Suspend Supported	Set to '1' if USB suspend is supported.
B27	Unconstrained Power	Set to '1' if unconstrained power is available.
B26	USB Communications Capable	Set to '1' if capable of <i>USB Communications</i> capable
B25	Dual-Role Data	Set to '1' for a <i>Dual-Role Data</i> device.
B24	Unchunked Extended Messages Supported	Set to '1' if <i>Unchunked Extended Messages</i> are supported.
B23	EPR Capable	Set to '1' if <i>EPR Capable</i> .
B22	Reserved	Reserved – Shall be set to zero.
B21...20	Peak Current	Peak Current value.
B19...10	Voltage	Voltage in 50mV units
B9...0	Maximum Current	Maximum current in 10mA units

6.4.1.2.1.1 Dual-Role Power

The **Dual-Role Power** bit **Shall** be set when the *Port* is *Dual-Role Power* capable i.e., supports the **PR_Swap** Message.

This is a **Static** capability which **Shall** remain fixed for a given device regardless of the device's present *Power Role*. If the **Dual-Role Power** bit is set to one in the **Source_Capabilities** Message the **Dual-Role Power** bit in the **Sink_Capabilities** Message **Shall** also be set to one. If the **Dual-Role Power** bit is set to zero in the **Source_Capabilities** Message the **Dual-Role Power** bit in the **Sink_Capabilities** Message **Shall** also be set to zero.

6.4.1.2.1.2 USB Suspend Supported

Prior to an *Explicit Contract* or when the **USB Communications Capable** bit is set to zero, the **USB Suspend Supported** flag is undefined and *Sinks* **Shall** follow the rules for suspend as defined in [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB4\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#). After an *Explicit Contract* has been *Negotiated*:

- If the **USB Suspend Supported** flag is set, then the *Sink* **Shall** follow the [\[USB 2.0\]](#), [\[USB 3.2\]](#) or [\[USB4\]](#) rules for suspend and resume. A *PDUSB Peripheral* **May** draw up to **pSnkSusp** during suspend; a *PDUSB Hub* **May** draw up to **pHubSusp** during suspend (see [Section 7.2.3, "Sink Standby"](#)).

- If the **USB Suspend Supported** flag is cleared, then the Sink **Shall Not** apply the **[USB 2.0]**, **[USB 3.2]** or **[USB4]** rules for suspend and **May** continue to draw the *Negotiated* power.

Note: When USB is suspended, the USB device state is also suspended.

Sinks **May** indicate to the Source that they would prefer to have the **USB Suspend Supported** flag cleared by setting the No USB Suspend flag in a **Request** Message (see [Section 6.4.2.5, "No USB Suspend"](#)).

6.4.1.2.1.3 Unconstrained Power

The **Unconstrained Power** bit **Shall** be set when an external source of power is available that is sufficient to adequately power the system while charging external devices, or when the device's primary function is to charge external devices.

To set the **Unconstrained Power** bit because of an external source, the external source of power **Should** be either:

- An AC Supply, e.g., a Charger, directly connected to the Sink.
- Or, in the case of a PDUSB Hub:
 - A PD Source with its **Unconstrained Power** bit set.
 - Multiple PD Sources all with their **Unconstrained Power** bits set.

6.4.1.2.1.4 USB Communications Capable

The **USB Communications Capable** bit **Shall** only be set for Sources capable of communication over the USB data lines (e.g., D+/- or SS Tx/Rx).

6.4.1.2.1.5 Dual-Role Data

The **Dual-Role Data** bit **Shall** be set when the Port is Dual-Role Data capable i.e., it supports the **DR_Swap** Message. This is a **Static** capability which **Shall** remain fixed for a given device regardless of the device's present Power Role or Data Role. If the **Dual-Role Data** bit is set to one in the **Source_Capabilities** Message the **Dual-Role Data** bit in the **Sink_Capabilities** Message **Shall** also be set to one. If the **Dual-Role Data** bit is set to zero in the **Source_Capabilities** Message the **Dual-Role Data** bit in the **Sink_Capabilities** Message **Shall** also be set to zero.

6.4.1.2.1.6 Unchunked Extended Messages Supported

The **Unchunked Extended Messages Supported** bit **Shall** be set when the Port can send and receive Extended Messages with **Data Size** > **MaxExtendedMsgLegacyLen** bytes in a single, *Unchunked Extended Message*.

6.4.1.2.1.7 EPR Mode Capable

The **EPR Capable** bit is a **Static** bit that **Shall** be set if the Source is designed to supply more than 100W and operate in *EPR Mode*.

When this bit is set, an *EPR Source*:

- Operating in *SPR Mode* **Shall** only send an **EPR_Source_Capabilities** Message in response to an **EPR_Get_Source_Cap** Message
- **May** only enter *EPR Mode* when the Cable and the Sink also report that they are *EPR Capable*.

6.4.1.2.1.8 Peak Current

The USB Power Delivery *Fixed Supply* is only required to deliver the amount of current requested in the **Operating Current** field (*IoC*) of an *RDO*. In some usages however, for example computer systems, where there are short bursts of activity, it might be desirable to overload the Source for short periods.

For example, when a computer system tries to maintain average power consumption, the higher the peak current, the longer the low current (see [Section 7.2.8, "Sink Peak Current Operation"](#)) period needed to maintain such average power. The **Peak Current** field allows a Source to *Advertise* this additional capability. This capability is intended for direct Port to Port connections only and **Shall Not** be offered to downstream Sinks via a Hub.

Every *Fixed Supply PDO* **Shall** contain a **Peak Current** field. Supplies that want to offer a set of overload *Capabilities* **Shall** Advertise this through the **Peak Current** field in the corresponding *Fixed Supply PDO* (see [Table 6.10, "Fixed Power Source Peak Current Capability"](#)). Supplies that do not support an overload capability **Shall** set these bits to 00b in the corresponding *Fixed Supply PDO*. Supplies that support an extended overload capability specified in the PeakCurrent1...3 fields of the **Source_Capabilities_Extended** Message (see [Section 6.5.1, "Source_Capabilities_Extended Message"](#)) **Shall** also set these bits to 00b. Sinks wishing to utilize these *Extended Capabilities* **Shall** first send the **Get_Source_Cap_Extended** Message to determine what *Capabilities*, if any are supported by the *Source*.

Table 6.10 Fixed Power Source Peak Current Capability

Bits 21...20	Description
00	Peak current equals <i>IoC</i> (default) or look at the Source_Capabilities_Extended Message (send Get_Source_Cap_Extended Message)
01	Overload <i>Capabilities</i> : 1. Peak current equals 150% <i>IoC</i> for 1ms @ 5% duty cycle (low current equals 97% <i>IoC</i> for 19ms) 2. Peak current equals 125% <i>IoC</i> for 2ms @ 10% duty cycle (low current equals 97% <i>IoC</i> for 18ms) 3. Peak current equals 110% <i>IoC</i> for 10ms @ 50% duty cycle (low current equals 90% <i>IoC</i> for 10ms)
10	Overload <i>Capabilities</i> : 1. Peak current equals 200% <i>IoC</i> for 1ms @ 5% duty cycle (low current equals 95% <i>IoC</i> for 19ms) 2. Peak current equals 150% <i>IoC</i> for 2ms @ 10% duty cycle (low current equals 94% <i>IoC</i> for 18ms) 3. Peak current equals 125% <i>IoC</i> for 10ms @ 50% duty cycle (low current equals 75% <i>IoC</i> for 10ms)
11	Overload <i>Capabilities</i> : 1. Peak current equals 200% <i>IoC</i> for 1ms @ 5% duty cycle (low current equals 95% <i>IoC</i> for 19ms) 2. Peak current equals 175% <i>IoC</i> for 2ms @ 10% duty cycle (low current equals 92% <i>IoC</i> for 18ms) 3. Peak current equals 150% <i>IoC</i> for 10ms @ 50% duty cycle (low current equals 50% <i>IoC</i> for 10ms)

6.4.1.2.2 Variable Supply (non-Battery) Power Data Object

[Table 6.11, "Variable Supply \(non-Battery\) PDO – Source"](#) describes a *Variable Supply (non-Battery)* (10b) *PDO* for a *Source*. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

The voltage fields **Shall** define the range that output voltage **Shall** fall within. This does not indicate the voltage that will be supplied, except it **Shall** fall within that range. The absolute voltage, including any voltage variation, **Shall Not** fall below the **Minimum Voltage** field value and **Shall Not** exceed the **Maximum Voltage** field value. The **Minimum Voltage** field value **Shall Not** be less than 80% of the **Maximum Voltage** field value.

Table 6.11 Variable Supply (non-Battery) PDO – Source

Bit(s)	Field	Description
B31...30	<i>Variable Supply</i>	01b - <i>Variable Supply (non-Battery) PDO</i>
B29...20	Maximum Voltage	Maximum voltage in 50mV units
B19...10	Minimum Voltage	Minimum voltage in 50mV units
B9...0	Maximum Current	Maximum current in 10mA units

6.4.1.2.3 Battery Supply Power Data Object

[Table 6.12, "Battery Supply PDO – Source"](#) describes a *Battery Supply* (01b) *PDO* for a *Source*. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

The voltage fields **Shall** represent the *Battery's* voltage range. The *Battery* **Shall** be capable of supplying the Power value over the entire voltage range. The absolute voltage, including any voltage variation, **Shall Not** fall below the **Minimum Voltage** field value and **Shall Not** exceed the **Maximum Voltage** field value.

Note: The *Battery Supply PDO* uses power instead of current.

The Sink **May** monitor the Battery voltage.

Table 6.12 Battery Supply PDO – Source

Bit(s)	Field	Description
B31...30	<i>Battery Supply</i>	10b - <i>Battery Supply PDO</i>
B29...20	<i>Maximum Voltage</i>	Maximum voltage in 50mV units
B19...10	<i>Minimum Voltage</i>	Minimum voltage in 50mV units
B9...0	<i>Maximum Allowable Power</i>	Maximum allowable power in 250mW units

6.4.1.2.4 Augmented Power Data Object (APDO)

The voltage fields define the output voltage range over which the power supply **Shall** be adjustable in 20mV steps in *SPR PPS Mode* and 100mV steps in both *SPR AVS Mode* and *EPR AVS Mode*. The **Maximum Current** field contains the current the Programmable Power Supply **Shall** be capable of delivering over the *Advertised* voltage range. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

6.4.1.2.4.1 SPR Programmable Power Supply APDO

[Table 6.13, "SPR Programmable Power Supply APDO – Source"](#) below describes the *SPR PPS* (1100b) *APDO* for a *Source* operating in *SPR Mode* and supplying 5V up to 21V.

Table 6.13 SPR Programmable Power Supply APDO – Source

Bit(s)	Field	Description
B31...30	<i>APDO</i>	11b – <i>Augmented Power Data Object (APDO)</i>
B29...28	<i>SPR PPS</i>	00b – <i>SPR PPS</i>
B27	PPS Power Limited	Set to '1' when <i>PPS Power Limited</i>
B26...25	Reserved	Reserved – Shall be set to zero.
B24...17	Maximum Voltage	Maximum voltage in 100mV increments
B16	Reserved	Reserved – Shall be set to zero.
B15...8	Minimum Voltage	Minimum voltage in 100mV increments
B7	Reserved	Reserved – Shall be set to zero.
B6...0	Maximum Current	Maximum current in 50mA increments

The *PPS APDO* is used primarily for *Sink Directed Charge* of a *Battery* in the *Sink*. When applying a current to the *Battery* greater than the cable supports, a high efficiency fixed voltage scaler **May** be used in the *Sink* to reduce the cable current.

6.4.1.2.4.1.1 PPS Power Limited

When the **PPS Power Limited** bit is set, the *SPR PPS Source* **Shall** operate in the same way as if the **PPS Power Limited** bit is clear (see [Section 7.1.4.2, "SPR Programmable Power Supply \(PPS\)"](#) with the below exception:

- **May** supply power that exceeds the *Source's* rated *PDP* within the **Optional** operating area in [Figure 7.7, "SPR PPS Constant Power"](#).

When the **PPS Power Limited** bit is cleared, the *SPR PPS Source* **Shall** deliver the **Maximum Current** field value up to the **Maximum Voltage** as *Advertised* in its *APDO*.

The *SPR PPS Source* **Shall Not** reject an *RDO* with an **Operating Current** field value that is less than or equal to the **Maximum Current** field value in the *APDO* even if the requested **Operating Current** field value is greater than the *Source's* *PDP*/requested Output voltage.

6.4.1.2.4.2 SPR Adjustable Voltage Supply APDO

[Table 6.14, "SPR Adjustable Voltage Supply APDO – Source"](#) below describes the *SPR AVS* (1110b) *APDO* for a *Source* operating in *SPR Mode* and supplying 9V up to 20V.

Table 6.14 SPR Adjustable Voltage Supply APDO – Source

Bit(s)	Field	Description
B31...30	<i>APDO</i>	11b – <i>Augmented Power Data Object (APDO)</i>
B29...28	<i>SPR AVS</i>	10b – <i>SPR AVS</i>
B27...26	Peak Current	Peak Current (see Table 6.10, "Fixed Power Source Peak Current Capability")
B25...20	Reserved	Reserved – <i>Shall</i> be set to zero.
B19...10	Maximum Current 15V	For 9V – 15V range: Maximum current in 10mA units equal to the Maximum Current field of the 15V <i>Fixed Supply PDO</i>
B9...0	Maximum Current 20V	For 15V – 20V range: Maximum current in 10mA units equal to the Maximum Current field of the 20V <i>Fixed Supply PDO</i> , set to 0 if the maximum voltage in the <i>SPR AVS</i> range is 15V.

6.4.1.2.4.2.1 Peak Current

The **Peak Current** field follows the same definition as for the **Peak Current** field (see [Section 6.4.1.2.1.8, "Peak Current"](#) and [Table 6.10, "Fixed Power Source Peak Current Capability"](#)).

6.4.1.2.4.3 EPR Adjustable Voltage Supply APDO

[Table 6.15, "EPR Adjustable Voltage Supply APDO – Source"](#) below describes the *EPR AVS* (1101b) *APDO* for a *Source* operating in *EPR Mode* and supplying 15V up to 48V.

Table 6.15 EPR Adjustable Voltage Supply APDO – Source

Bit(s)	Field	Description
B31...30	<i>APDO</i>	11b – <i>Augmented Power Data Object (APDO)</i>
B29...28	<i>EPR AVS</i>	01b – <i>EPR AVS</i>
B27...26	Peak Current (Source EPR AVS)	Peak Current (see Table 6.16, "EPR AVS Power Source Peak Current Capability")
B25...17	Maximum Voltage	Maximum voltage in 100mV increments
B16	Reserved	Reserved – <i>Shall</i> be set to zero.
B15...8	Minimum Voltage	Minimum voltage in 100mV increments
B7...0	PDP	<i>PDP</i> in 1W increments

6.4.1.2.4.3.1 PDP

The **PDP** field **Shall** contain the *AVS Port's PDP*.

See [Section 10.2.3.3, "Optional Normative Extended Power Range \(EPR\)"](#) and [Figure 10.6, "Valid EPR AVS Operating Region"](#) for more information regarding how *PDP* in the *AVS APDO* relates to maximum available current.

6.4.1.2.4.3.2 Peak Current

The USB Power Delivery *EPR AVS* is only required to deliver the amount of current requested in the **Operating Current** field (*IoC*) of an *AVS RDO*. In some usages however, for example computer systems, where there are short bursts of activity, it might be desirable to overload the *Source* for short periods.

For example, when a computer system tries to maintain average power consumption, the higher the peak current, the longer the low current period needed to maintain such average power (see [Section 7.2.8, "Sink Peak Current Operation"](#)). The **Peak Current (Source EPR AVS)** field allows a *Source* to *Advertise* this additional capability. This

capability is intended for direct *Port to Charger* connections only and **Shall Not** be offered to downstream *Sinks* via a *Hub*.

Every *EPR AVS APDO* **Shall** contain a **Peak Current (Source EPR AVS)** field. Supplies that want to offer a set of overload *Capabilities* **Shall** Advertise this through the **Peak Current (Source EPR AVS)** field in the corresponding *EPR AVS APDO* (see [Table 6.16, "EPR AVS Power Source Peak Current Capability"](#)). Supplies that do not support an overload capability **Shall** set these bits to 00b in the corresponding *EPR AVS APDO*. Supplies that support an extended overload capability specified in the PeakCurrent1...3 fields of the **Source_Capabilities_Extended** Message (see [Section 6.5.1, "Source_Capabilities_Extended Message"](#)) **Shall** set these bits to 00b. *Sinks* wishing to utilize these *Extended Capabilities* **Shall** first send a **Get_Source_Cap_Extended** Message to determine what *Capabilities*, if any are supported by the *Source*.

Table 6.16 EPR AVS Power Source Peak Current Capability

Bits 21...20	Description
00	Peak current equals <i>IoC</i> (default) or look at the Source_Capabilities_Extended Message (send Get_Source_Cap_Extended Message)
01	Overload <i>Capabilities</i> : 1. Peak current equals 150% <i>IoC</i> for 1ms @ 5% duty cycle (low current equals 97% <i>IoC</i> for 19ms) 2. Peak current equals 125% <i>IoC</i> for 2ms @ 10% duty cycle (low current equals 97% <i>IoC</i> for 18ms) 3. Peak current equals 110% <i>IoC</i> for 10ms @ 50% duty cycle (low current equals 90% <i>IoC</i> for 10ms)
10	Overload <i>Capabilities</i> : 1. Peak current equals 200% <i>IoC</i> for 1ms @ 5% duty cycle (low current equals 95% <i>IoC</i> for 19ms) 2. Peak current equals 150% <i>IoC</i> for 2ms @ 10% duty cycle (low current equals 94% <i>IoC</i> for 18ms) 3. Peak current equals 125% <i>IoC</i> for 10ms @ 50% duty cycle (low current equals 75% <i>IoC</i> for 10ms)
11	Overload <i>Capabilities</i> : 1. Peak current equals 200% <i>IoC</i> for 1ms @ 5% duty cycle (low current equals 95% <i>IoC</i> for 19ms) 2. Peak current equals 175% <i>IoC</i> for 2ms @ 10% duty cycle (low current equals 92% <i>IoC</i> for 18ms) 3. Peak current equals 150% <i>IoC</i> for 10ms @ 50% duty cycle (low current equals 50% <i>IoC</i> for 10ms)

6.4.1.3 Sink Power Data Objects

This section lists the types of *PDOs* a *Sink* can use in an *SPR* or *EPR Capabilities Message*.

6.4.1.3.1 Sink Fixed Supply Power Data Object

[Table 6.17, "Fixed Supply PDO – Sink"](#) describes the *Sink Fixed Supply* (00b) *PDO*. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply. The *Sink* **shall** set the **Voltage** field to its required voltage and the **Operational Current** field to its required operating current. Required operating current is defined as the amount of current a given device needs to be functional. This value could be the maximum current the *Sink* will ever require or could be sufficient to operate the *Sink* in one of its modes of operation.

Since all USB *Consumers* support **vSafe5V**, the required **vSafe5V Fixed Supply Power Data Object** is also used to convey additional information that is returned in bits 29 through 20. All other *Fixed Supply Power Data Objects* **shall** set bits 29...20 to zero.

For a *Sink* requiring no power from the *Source*, the **Voltage** field **shall** be set to 5V and the **Operational Current** field **shall** be set to 0mA.

Table 6.17 Fixed Supply PDO – Sink

Bit(s)	Field	Description	
B31...30	<i>Fixed Supply</i>	00b - <i>Fixed Supply PDO</i>	
B29	<i>Dual-Role Power</i>	Set to ‘1’ if <i>Dual-Role Power</i> supported	
B28	<i>Higher Capability</i>	Set to ‘1’ if <i>Higher Capability</i> supported	
B27	<i>Unconstrained Power</i>	Set to ‘1’ if <i>Unconstrained Power</i> supported	
B26	<i>USB Communications Capable</i>	Set to ‘1’ if <i>USB Communications Capable</i>	
B25	<i>Dual-Role Data</i>	<i>Dual-Role Data</i>	
B24...23	<i>Fast Role Swap required USB Type-C Current</i>	<i>Fast Role Swap</i> required <i>USB Type-C</i> current (see also [USB Type-C 2.4f]):	
		Value	Description
		00b	<i>Fast Role Swap</i> not supported (default)
		01b	Default <i>USB Port</i>
		10b	1.5A@5V
		11b	3.0A@5V
B22...20	<i>Reserved</i>	<i>Reserved</i> – <i>Shall</i> be set to zero.	
B19...10	<i>Voltage</i>	Voltage in 50mV units	
B9...0	<i>Operational Current</i>	Operational current in 10mA units	

6.4.1.3.1.1 Dual-Role Power

The **Dual-Role Power** bit **shall** be set when the *Port* is *Dual-Role Power* capable i.e., supports the **PR_Swap** Message. This is a **Static** capability which **shall** remain fixed for a given device regardless of the device's present *Power Role*. If the **Dual-Role Power** bit is set to one in the *Source_Capabilities* Message the **Dual-Role Power** bit in the *Sink_Capabilities* Message **shall** also be set to one. If the **Dual-Role Power** bit is set to zero in the *Source_Capabilities* Message the **Dual-Role Power** bit in the *Sink_Capabilities* Message **shall** also be set to zero.

6.4.1.3.1.2 Higher Capability

In the case that the *Sink* needs more than **vSafe5V** (e.g., 15V) to provide full functionality, then the **Higher Capability** bit **shall** be set.

6.4.1.3.1.3 Unconstrained Power

The **Unconstrained Power** bit **shall** be set when an external source of power is available that is sufficient to adequately power the system while charging external devices, or when the device's primary function is to charge external devices.

To set the **Unconstrained Power** bit because of an external source, the external source of power **Should** be either:

- An AC Supply, e.g., a Charger, directly connected to the Sink.
- Or, in the case of a PDUSB Hub:
 - A PD Source with its **Unconstrained Power** bit set.
 - Multiple PD Sources all with their **Unconstrained Power** bits set.

6.4.1.3.1.4 USB Communications Capable

The **USB Communications Capable** bit **Shall** only be set for Sinks capable of communication over the USB data lines (e.g., D+/- or SS Tx/Rx).

6.4.1.3.1.5 Dual-Role Data

The **Dual-Role Data** bit **Shall** be set when the Port is Dual-Role Data capable i.e., it supports the **DR_Swap** Message. This is a **Static** capability which **Shall** remain fixed for a given device regardless of the device's present Power Role or Data Role. If the **Dual-Role Data** bit is set to one in the **Source_Capabilities** Message the **Dual-Role Data** bit in the **Sink_Capabilities** Message **Shall** also be set to one. If the **Dual-Role Data** bit is set to zero in the **Source_Capabilities** Message the **Dual-Role Data** bit in the **Sink_Capabilities** Message **Shall** also be set to zero.

6.4.1.3.1.6 Fast Role Swap USB Type-C Current

The **Fast Role Swap required USB Type-C Current** field **Shall** indicate the current level the Sink will require after a Fast Role Swap has been performed.

The Initial Source **Shall Not** transmit a Fast Role Swap Request if the **Fast Role Swap required USB Type-C Current** field is set to zero.

Initially when the New Source applies **vSafe5V** it will have R_d asserted but **Shall** provide the USB Type-C current indicated by the New Sink in this field. If the New Source is not able to supply this level of current, it **Shall Not** perform a Fast Role Swap. When R_p is asserted by the New Source during the Fast Role Swap AMS (see [Section 6.3.19, "FR_Swap Message"](#)), the value of USB Type-C current indicated by R_p **Shall** be the same or greater than that indicated in the **Fast Role Swap required USB Type-C Current** field.

6.4.1.3.2 Variable Supply (non-Battery) Power Data Object

[Table 6.18, "Variable Supply \(non-Battery\) PDO – Sink"](#) describes a Variable Supply (non-Battery) (10b) PDO used by a Sink. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

The voltage fields **Shall** be set to the output voltage range that the Sink requires to operate. The **Operational Current** field **Shall** be set to the operational current that the Sink requires at the given voltage range. The absolute voltage, including any voltage variation, **Shall Not** fall below the **Minimum Voltage** field value and **Shall Not** exceed the **Maximum Voltage** field value. Required operating current is defined as the amount of current a given device needs to be functional. This value could be the maximum current the Sink will ever require or could be sufficient to operate the Sink in one of its modes of operation.

Table 6.18 Variable Supply (non-Battery) PDO – Sink

Bit(s)	Field	Description
B31...30	Variable Supply	01b - Variable Supply (non-Battery) PDO
B29...20	Maximum Voltage	Maximum voltage in 50mV units
B19...10	Minimum Voltage	Minimum voltage in 50mV units
B9...0	Operational Current	Operational current in 10mA units

6.4.1.3.3 Battery Supply Power Data Object

[Table 6.19, "Battery Supply PDO – Sink"](#) describes a Battery Supply (01b) PDO used by a Sink. See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

The voltage fields **Shall** be set to the output voltage range that the *Sink* requires to operate. The **Operational Power** field **Shall** be set to the operational power that the *Sink* requires at the given voltage range. The absolute voltage, including any voltage variation, **Shall Not** fall below the **Minimum Voltage** field value and **Shall Not** exceed the **Maximum Voltage** field value.

Note: Only the *Battery Supply PDO* uses power instead of current.

Required operating power is defined as the amount of power a given device needs to be functional. This value could be the maximum power the *Sink* will ever require or could be sufficient to operate the *Sink* in one of its modes of operation.

Table 6.19 Battery Supply PDO – Sink

Bit(s)	Field	Description
B31...30	<i>Battery Supply</i>	10b - <i>Battery Supply PDO</i>
B29...20	Maximum Voltage	Maximum voltage in 50mV units
B19...10	Minimum Voltage	Minimum voltage in 50mV units
B9...0	Operational Power	Operational Power in 250mW units

6.4.1.3.4 Augmented Power Data Objects

See [Section 7.1.3, "Types of Sources"](#) for the electrical requirements of the power supply.

The Maximum and Minimum voltage fields **Shall** be set to the output voltage range that the *Sink* requires to operate.

6.4.1.3.4.1 SPR Programmable Power Supply APDO

[Table 6.20, "SPR Programmable Power Supply APDO – Sink"](#) below describes a *SPR PPS APDO* for a *Sink* operating in *SPR Mode* and consuming 21V or less. The **Maximum Current** field **Shall** be set to the maximum current the *Sink* requires over the voltage range. The maximum current is defined as the maximum amount of current the device needs to fully support its function (e.g., *Sink Directed Charge*).

Table 6.20 SPR Programmable Power Supply APDO – Sink

Bit(s)	Field	Description
B31...30	<i>APDO</i>	11b – <i>Augmented Power Data Object (APDO)</i>
B29...28	<i>SPR PPS</i>	00b – <i>SPR PPS</i>
B27...25	Reserved	Reserved – Shall be set to zero.
B24...17	Maximum Voltage	Maximum voltage in 100mV increments
B16	Reserved	Reserved – Shall be set to zero.
B15...8	Minimum Voltage	Minimum voltage in 100mV increments
B7	Reserved	Reserved – Shall be set to zero.
B6...0	Maximum Current	Maximum current in 50mA increments

6.4.1.3.4.2

SPR Adjustable Voltage Supply APDO

[Table 6.21, "SPR Adjustable Voltage Supply APDO – Sink"](#) below describes the *SPR AVS* (1110b) *APDO* for a *Sink* operating in *SPR AVS Mode*. The **Maximum Current 15V**/**Maximum Current 20V** fields in the *SPR AVS APDO* for the *Sink* is defined as the maximum current the device needs to fully support its function.

Table 6.21 SPR Adjustable Voltage Supply APDO – Sink

Bit(s)	Field	Description
B31...30	<i>APDO</i>	11b – <i>Augmented Power Data Object (APDO)</i>
B29...28	<i>SPR AVS</i>	10b – <i>SPR AVS</i>
B27...20	Reserved	Reserved – Shall be set to zero.
B19...10	Maximum Current 15V	For 9V – 15V range: Maximum current in 10mA units equal to the Maximum Current field of the <i>15V Fixed Supply PDO</i>
B9...0	Maximum Current 20V	For 15V – 20V range: Maximum Current in 10mA units equal to the Maximum Current field of the <i>20V Fixed Supply PDO</i> , set to 0 if the Maximum voltage in the <i>SPR AVS</i> range is 15V.

6.4.1.3.4.3

EPR Adjustable Voltage Supply APDO

[Table 6.22, "EPR Adjustable Voltage Supply APDO – Sink"](#) below describes a *EPR AVS APDO* for a *Sink* operating in *EPR AVS Mode*. The **PDP** field in the *EPR AVS APDO* for the *Sink* is defined as the *PDP* the device needs to fully support its function.

Table 6.22 EPR Adjustable Voltage Supply APDO – Sink

Bit(s)	Field	Description
B31...30	<i>APDO</i>	11b – <i>Augmented Power Data Object (APDO)</i>
B29...28	<i>EPR AVS</i>	01b – <i>EPR AVS</i>
B27...26	Reserved	Reserved – Shall be set to zero.
B25...17	Maximum Voltage	Maximum voltage in 100mV increments
B16	Reserved	Reserved – Shall be set to zero.
B15...8	Minimum Voltage	Minimum voltage in 100mV increments
B7...0	PDP	<i>PDP</i> in 1W increments

6.4.1.4

SPR Capabilities Message Construction

An *SPR Capabilities Message* (**Source Capabilities Message** or **Sink Capabilities Message**) **Shall** have at least one *Power Data Object* for **vSafe5V**. The *SPR Capabilities Message* **Shall** also contain the sending *Port*'s information followed by up to 6 additional *Power Data Objects*. *Power Data Objects* in an *SPR Capabilities Message* **Shall** be sent in the following order:

- 1) The **vSafe5V Fixed Supply PDO** **Shall** always be the first (*A*)*PDO*.
- 2) The remaining *Fixed Supply PDOs*, if present, **Shall** be sent in voltage order; lowest to highest.
- 3) The *Battery Supply PDOs* if present **Shall** be sent in Minimum voltage order; lowest to highest.
- 4) The *Variable Supply (non-Battery) PDOs*, if present, **Shall** be sent in Minimum voltage order; lowest to highest.
- 5) The *SPR AVS APDO*, if present, **Shall** be sent.
- 6) The *Programmable Power Supply APDOs*, if present, **Shall** be sent in Maximum voltage order, lowest to highest.

Note: The *EPR Capabilities Message* construction is defined in [Section 6.5.15.1, "EPR Capabilities Message Construction"](#).

[Figure 6.12, "SPR Capabilities Message Construction"](#) describes the construction of an *SPR Capabilities Message*. The Message will always have at least one *Fixed Supply 5V PDO* and may have up to six more *PDOs* depending on the *Source Capabilities*.

Figure 6.12 SPR Capabilities Message Construction

Header 2 bytes	PDO 1	PDO 2	PDO 3	PDO 4	PDO 5	PDO 6	PDO 7
	001b	010b	011b	100b	101b	110b	111b

Figure 6.13 Example Capabilities Message with 2 Power Data Objects

Header No. of Data Objects = 2	Fixed 5V PDO	Fixed 9V PDO
-----------------------------------	-----------------	-----------------

In the 27W Source as shown in [Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"](#), the **Number of Data Objects** field is 2: **vSafe5V** plus one other voltage.

Power Data Objects (PDO) and *Augmented Power Data Objects (APDO)* are identified by the *Message Header's Message Type* field. They are used to form *SPR Capabilities Messages*.

6.4.1.5 SPR Source Capabilities Message

Sources send a **Source_Capabilities** Message either as part of advertising *Port Capabilities*, or in response to a **Get_Source_Cap** Message. See [Section 6.5.15.2, "EPR Source Capabilities Message"](#) for information about *EPR Source Capabilities Messages*.

Following a *Hard Reset*, a power-on event or plug insertion event, a *Source Port* **Shall** send a **Source_Capabilities** Message after every **SourceCapabilityTimer** timeout as an *Advertisements* that **Shall** be interpreted by the *Sink Port* on Attachment. The *Source* **Shall** continue sending a minimum of **nCapsCount** **Source_Capabilities** Messages until a **GoodCRC** Message is received.

Additionally, a **Source_Capabilities** Message **Shall** only be sent by a *Port* in the following cases:

- By the *Source Port* from the **PE_SRC_Ready** state upon a change in its ability to supply power to this *Port*.
- By a *Source Port* or *Dual-Role Power Port* in response to a **Get_Source_Cap** Message.
- **Optionally** by a *Source Port* from the **PE_SRC_Ready** state when available power in a multi-*Port* system changes, even if the *Source Capabilities* for this *Port* have not changed.

A *Source Port* **Shall** report its *Capabilities* in a series of 32-bit *Power Data Objects* (see [Table 6.7, "Power Data Object"](#)) as part of a **Source_Capabilities** Message (see [Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"](#)). *Power Data Objects* are used to convey a *Source Port's Capabilities* to provide power including *Dual-Role Power ports* presently operating as a *Sink*.

Each *Power Data Object* **Shall** describe a specific *Source* capability such as a *Battery* (e.g., 2.8-4.1V) or a *Fixed Supply* (e.g., 15V) at a maximum allowable current. The **Number of Data Objects** field in the *Message Header* **Shall** define the number of *Power Data Objects* that follow the *Message Header* in a *Data Message*. All *Sources* **Shall** minimally offer one *Power Data Object* that reports **vSafe5V**. A *Source* **Shall Not** offer multiple *Power Data Objects* of the same type (*Fixed Supply*, *Variable Supply*, *Battery Supply*) and the same voltage but **Shall** instead offer one *Power Data Object* with the highest available current for that *Source* capability and voltage.

Sinks with Accessory Support do not source *V_{BUS}* (see [\[USB Type-C 2.4\]](#)). Sinks with Accessory Support are still considered *Sources* when sourcing *V_{CONN}* to an Accessory even though *V_{BUS}* is not applied; in this case they **Shall** Advertise *vSafe5V* with the **Maximum Current** field set to 0mA in the first *Power Data Object*. The main purpose of this is to enable the Sink with Accessory Support to get into the **PE_SRC_Ready** State to enter an *Alternate Mode*.

A Sink in *SPR Mode* **Shall** evaluate every *Source_Capabilities* Message it receives and **Shall** respond with a *Request* Message. If its power consumption exceeds the *Source Capabilities* it **Shall** Re-negotiate so as not to exceed the *Source's* most recently Advertised *Capabilities*.

A Sink, in *SPR Mode*, in an *Explicit Contract* with a *PPS APDO*, **Shall** periodically re-request the *PPS APDO* at least every *tPPSRequest* until either:

- The Sink requests something other than *PPS APDO*.
- There is a *Power Role Swap*.
- There is a *Hard Reset*.
- There is *Error Recovery*.

A Sink in *EPR Mode* that receives a *Source_Capabilities* Message in response to a *Get_Source_Cap* Message **Shall Not** respond with a *Request* Message. If a Sink in *EPR Mode* receives a *Source_Capabilities* Message, not in response to a *Get_Source_Cap* Message, the Sink **Shall** initiate a *Hard Reset*.

A Source that has accepted a *Request* Message with a Programmable *RDO* **Shall** issue *Hard Reset* Signaling if it has not received a *Request* Message with a Programmable *RDO* within *tPPSTimeout*. The Source **Shall** discontinue this behavior after:

- Receiving a *Request* Message with a *Fixed Supply*, *Variable Supply* or *Battery Supply* *RDO*.
- There is a *Power Role Swap*.
- There is a *Hard Reset*.
- There is *Error Recovery*.

6.4.1.6 SPR Sink Capabilities Message

Sinks send a *Sink_Capabilities* Message (see [Section 6.4.2, "Request Message"](#)) in response to a *Get_Sink_Cap* Message. See [Section 6.5.15.3, "EPR Sink Capabilities Message"](#) for more information about the *Capabilities* Message.

A USB Power Delivery capable Sink, upon detecting *vSafe5V* on *V_{BUS}* and after a *SinkWaitCapTimer* timeout without seeing a *Source_Capabilities* Message, **Shall** send a *Hard Reset*. If the Attached Source is USB Power Delivery capable, it responds by sending *Source_Capabilities* Messages thus allowing power *Negotiations* to begin.

A Sink Port **Shall** report power levels it is able to operate at in a series of 32-bit *Power Data Objects* (see [Section Table 6.7, "Power Data Object"](#)). These are returned as part of a *Sink_Capabilities* Message in response to a *Get_Sink_Cap* Message (see [Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"](#)). This is similar to that used for *Source Port Capabilities* with equivalent *Power Data Objects* for *Fixed Supply*, *Variable Supply* and *Battery Supply* as defined in this section. *Power Data Objects* are used to convey the Sink Port's operational power requirements including *Dual-Role Power Ports* presently operating as a *Source*.

Each *Power Data Object* **Shall** describe a specific Sink operational power level, such as a *Battery Supply* (e.g., 2.8-4.1V) or a *Fixed Supply* (e.g., 15V). The **Number of Data Objects** field in the *Message Header* **Shall** define the number of *Power Data Objects* that follow the *Message Header* in a *Data Message*.

All Sinks **Shall** minimally offer one *Power Data Object* with a power level at which the Sink can operate. A Sink **Shall Not** offer multiple *Power Data Objects* of the same type (*Fixed Supply*, *Variable Supply*, *Battery Supply*) and the same voltage but **Shall** instead offer one *Power Data Object* with the highest available current for that Sink capability and voltage.

All Sinks **Shall** include one *Power Data Object* that reports *vSafe5V* even if they require additional power to operate fully. In the case where additional power is required for full operation the Higher Capability bit **Shall** be set.

6.4.1.6.1 Use by Dual-Role Power devices

Dual-Role Power devices send a **Source_Capabilities** Message (see [Section 6.4.1.5, "SPR Source Capabilities Message"](#)) as part of advertising *Port Capabilities* when operating in *Source* role. *Dual-Role Power* devices send a **Source_Capabilities** Message in response to a **Get_Source_Cap** Message regardless of their present operating role. Similarly *Dual-Role Power* devices send a **Sink_Capabilities** Message (see [Section 6.4.1.6, "SPR Sink Capabilities Message"](#)) in response to a **Get_Sink_Cap** Message regardless of their present operating role.

6.4.1.6.2 Management of the Power Reserve

This section has been removed. Refer to [Section 8.2.5, "Managing Power Requirements"](#).

6.4.2 Request Message

A **Request** Message **Shall** be sent by a Sink to request power during the request phase of an SPR power Negotiation. The Request Data Object **Shall** be returned by the Sink making a request for power. It **Shall** be sent in response to the most recent **Source_Capabilities** Message (see [Section 8.3.2.2, "Power Negotiation"](#)) when in SPR Mode. A **Request** Message **Shall** return one and only one Sink Request Data Object that **Shall** identify the Power Data Object being requested.

The Source **Shall** respond to a **Request** Message with an **Accept** Message, a **Wait** Message or a **Reject** Message (see [Section 6.9, "Accept, Reject and Wait"](#)).

The **Request** Message includes the requested power level. For example, if the **Source_Capabilities** Message includes a Fixed Supply PDO that offers 9V @ 1.5A and if the Sink only wants 9V @ 0.5A, it will set the **Operating Current** field to 50 (i.e., 10mA * 50 = 0.5A).

The request uses a different format depending on the kind of power requested.

- The Fixed Supply Power Data Object and Variable Supply Power Data Object share a common format shown in [Table 6.23, "Fixed and Variable Request Data Object"](#).
- The Battery Supply Power Data Object uses the format shown in [Table 6.24, "Battery Request Data Object"](#).
- The PPS Request Data Object's format is shown in [Table 6.25, "PPS Request Data Object"](#).
- The AVS Request Data Object's format is shown in [Table 6.26, "AVS Request Data Object"](#).

The Request Data Objects are also used by the **EPR_Request** Message when operating in EPR Mode. See [Section 6.4.9, "EPR_Request Message"](#) for information about the use of the **EPR_Request** Message.

A Source operating in EPR Mode that receives a **Request** Message **Shall** initiate a Hard Reset.

Table 6.23 Fixed and Variable Request Data Object

Bit(s)	Field	Description
B31...28	Object Position	Object position (0000b and 1110b...1111b are Reserved and Shall Not be used)
B27	Giveback	GiveBack flag = 0 - Deprecated and Shall be set to zero.
B26	Capability Mismatch	Set to '1' for a <i>Capabilities Mismatch</i>
B25	USB Communications Capable	Set to '1' if <i>USB Communications Capable</i>
B24	No USB Suspend	Set to '1' if requesting No USB Suspend
B23	Unchunked Extended Messages Supported	Set to '1' if <i>Unchunked Extended Messages Supported</i>
B22	EPR Capable	Set to '1' if <i>EPR Capable</i>
B21...20	Reserved	Reserved – Shall be set to zero.
B19...10	Operating Current	Operating current in 10mA units
B9...0	Maximum Operating Current	Maximum Operating current 10mA units

Table 6.24 Battery Request Data Object

Bit(s)	Field	Description
B31...28	<i>Object Position</i>	Object position (0000b and 1110b...1111b are Reserved and Shall Not be used)
B27	<i>Giveback</i>	GiveBack flag = 0- Deprecated and Shall be set to zero.
B26	<i>Capability Mismatch</i>	Set to '1' for a <i>Capabilities Mismatch</i>
B25	<i>USB Communications Capable</i>	Set to '1' if <i>USB Communications Capable</i>
B24	<i>No USB Suspend</i>	Set to '1' if requesting No USB Suspend
B23	<i>Unchunked Extended Messages Supported</i>	Set to '1' if <i>Unchunked Extended Messages Supported</i>
B22	<i>EPR Capable</i>	Set to '1' if <i>EPR Capable</i>
B21...20	Reserved	Reserved – Shall be set to zero.
B19...10	<i>Operating Power</i>	Operating Power in 250mW units
B9...0	<i>Maximum Operating Power</i>	Maximum Operating Power in 250mW units

Table 6.25 PPS Request Data Object

Bit(s)	Field	Description
B31...28	<i>Object Position</i>	Object position (0000b and 1110b...1111b are Reserved and Shall Not be used)
B27	Reserved	Reserved – Shall be set to zero.
B26	<i>Capability Mismatch</i>	Set to '1' for a <i>Capabilities Mismatch</i>
B25	<i>USB Communications Capable</i>	Set to '1' if <i>USB Communications Capable</i>
B24	<i>No USB Suspend</i>	Set to '1' if requesting No USB Suspend
B23	<i>Unchunked Extended Messages Supported</i>	Set to '1' if <i>Unchunked Extended Messages Supported</i>
B22	<i>EPR Capable</i>	Set to '1' if <i>EPR Capable</i>
B21	Reserved	Reserved – Shall be set to zero.
B20...9	<i>Output Voltage</i>	Output voltage in 20mV units.
B8...7	Reserved	Reserved – Shall be set to zero.
B6...0	<i>Operating Current</i>	Operating current 50mA units.

Table 6.26 AVS Request Data Object

Bit(s)	Field	Description
B31...28	<i>Object Position</i>	Object position (0000b and 1110b...1111b are Reserved and Shall Not be used)
B27	Reserved	Reserved – Shall be set to zero.
B26	<i>Capability Mismatch</i>	Set to '1' for a <i>Capabilities Mismatch</i>
B25	<i>USB Communications Capable</i>	Set to '1' if <i>USB Communications Capable</i>
B24	<i>No USB Suspend</i>	Set to '1' if requesting No USB Suspend
B23	<i>Unchunked Extended Messages Supported</i>	Set to '1' if <i>Unchunked Extended Messages Supported</i>
B22	<i>EPR Capable</i>	Set to '1' if <i>EPR Capable</i>
B21	Reserved	Reserved – Shall be set to zero.
B20...9	<i>Output Voltage</i>	Output voltage in 25mV units, the least two significant bits Shall be set to zero making the effective voltage step size 100mV.
B8...7	Reserved	Reserved – Shall be set to zero.
B6...0	<i>Operating Current</i>	Operating current 50mA units.

6.4.2.1 Object Position

The value in the **Object Position** field **Shall** indicate which object in the **Source_Capabilities** Message or **EPR_Source_Capabilities** Message the **RDO** refers to. The value 0001b always indicates the 5V Fixed Supply **PDO** as it is the first object following the **Source_Capabilities** Message's Message Header or **EPR_Source_Capabilities** Message's Extended Message Header. The number 0010b refers to the next **PDO** and so forth.

The **Object Position** field values 0001b...0111b **Shall** only be used to refer to **SPR (A)PDOs**. **SPR (A)PDOs** **May** be requested by either a **Request** or an **EPR_Request** Message. Object positions 1000b...1011b **Shall** only be used to refer to **EPR (A)PDOs**. **EPR (A)PDOs** **Shall** only be requested by an **EPR_Request** Message. If the **Object Position** field in a **Request** Message contains a value greater than 0111b, the **Source** **Shall** send **Hard Reset** Signaling.

6.4.2.2 GiveBack Flag (Deprecated)

The **Giveback** flag has been **Deprecated** and **Shall** be set to zero.

6.4.2.3 Capability Mismatch

A **Capabilities Mismatch** occurs when the **Source** cannot satisfy the **Sink**'s power requirements based on the **Source Capabilities** it has offered. In this case the **Sink** **Shall** make a **Valid** request from the offered **Source Capabilities** and **Shall** set the **Capability Mismatch** bit (see [Section 8.2.5.2, "Power Capability Mismatch"](#)). When a **Capabilities Mismatch** condition does not exist, the **Sink** **Shall Not** set the **Capability Mismatch** bit.

When a **Sink** returns a **Request Data Object** with the **Capability Mismatch** bit set in response to a **Source Capabilities** Message, it indicates that it wants more power than the **Source** is currently offering. This can be due to either a specific voltage that is not being offered or there is not sufficient current for the voltages that are being offered.

Sources whose **Port Reported PDP** is less than their **Port Present PDP** (see [Section 6.4.11, "Source_Info Message"](#)) **Shall** respond to the Requests with the **Capability Mismatch** bit set as follows. The **Source** within **tCapabilitiesMismatchResponse** of the **PS_RDY** Message **Shall** send a new **Source Capabilities** Message that offers either:

- 1) The set of **Source Capabilities** to minimally satisfy the **Sink**'s requirements based on what it actually requires for full operation by evaluating the:
 - a) **Sink_Capabilities_Extended** Message (if supported by the **Sink**) and/or
 - b) **Sink_Capabilities** or **EPR_Sink_Capabilities** Message.
- 2) The set of **Source Capabilities** the **Source** can supply at this time based on the **Port Present PDP**.

To prevent looping, **Sources** **Should Not** send a new **Source Capabilities** Message in response to subsequent **Request** Message with the **Capability Mismatch** flag set until its **Port Present PDP** changes.

Once a **Guaranteed Capability Source** that has responded to a **Capability Mismatch**, it **Shall Not** subsequently send out another **Source Capabilities** Message at a lower **PDP** unless the power required by the **Sink** (as indicated in its **Sink Capabilities** Message or **Sink_Capabilities_Extended** Message) has also been reduced. **Sources** wishing to manage their power **May** periodically check the **Sink Capabilities** Message or **Sink_Capabilities_Extended** Message to determine whether these have changed.

Note: A **Source Capabilities** Message refers to a **Source_Capabilities** Message or an **EPR_Source_Capabilities** Message, and a **Sink Capabilities** Message refers to a **Sink_Capabilities** Message or **EPR_Sink_Capabilities** Message, **Request** refers to a **Request** Message or **EPR_Request** depending on operating mode.

In this context a **Valid** Request Message means the following:

- The **Object Position** field **Shall** contain a reference to an object that was present in the last received **Source Capabilities** Message.
- The **Operating Current/Operating Power** field **Shall** contain a value which is less than or equal to the maximum current/power offered by the selected (A)PDO the **Source Capabilities** Message.

6.4.2.4 USB Communications Capable

The **USB Communications Capable** flag **Shall** be set to one when the *Sink* has USB data lines and is capable of communicating using either **[USB 2.0]**, **[USB 3.2]** or **[USB4]** protocols. The **USB Communications Capable** flag **Shall** be set to zero when the *Sink* does not have USB data lines or is otherwise incapable of communicating using either **[USB 2.0]**, **[USB 3.2]** or **[USB4]** protocols. This is used by the *Source* to determine operation in certain cases such as USB suspend. If the **USB Communications Capable** flag has been set to zero by a *Sink*, then the *Source* needs to be aware that USB Suspend rules cannot be observed by the *Sink*.

6.4.2.5 No USB Suspend

The **No USB Suspend** flag **May** be set by the *Sink* to indicate to the *Source* that this device is requesting to continue its *Explicit Contract* during USB Suspend. *Sinks* setting this flag typically have functionality that can use power for purposes other than *USB Communication* e.g., for charging a *Battery*.

The *Source* uses this flag to evaluate whether it **Should** re-issue the **Source_Capabilities** Message with the **USB Suspend Supported** flag cleared.

6.4.2.6 Unchunked Extended Messages Supported

The **Unchunked Extended Messages Supported** bit **Shall** be set when the *Port* can send and receive *Extended Messages* with **Data Size** > **MaxExtendedMsgLegacyLen** bytes in a single, *Unchunked Extended Message*.

6.4.2.7 EPR Mode Capable

The **EPR Capable** bit **Shall** indicate whether or not the *Sink* is capable of operating in *EPR Mode*. When the *Sink's* ability to operate in *EPR Mode* changes, it **Shall** send a new **Request** Message with the updated **EPR Capable** bit set in the *RDO*.

6.4.2.8 Operating Current

The **Operating Current** field in the *Request Data Object* **Shall** be set to the highest current the *Sink* will draw during the *Explicit Contract*. A new **Request** Message or **EPR_Request** Message, with an updated **Operating Current** value, **Shall** be issued whenever the *Sink's* power needs change.

The **Operating Current** field in the *SPR Programmable Request Data Object* is used in addition by the *Sink* to request the *Source* for the *Current Limit* level it needs. When the request is accepted the *Source's* output current supplied into any load **Shall** be less than or equal to the **Operating Current** value. When the *Sink* attempts to consume more current, the *Source* **Shall** reduce the output voltage so as not to exceed the **Operating Current** value.

The **Operating Current** field in the *AVS Request Data Object* **Shall** be set to the highest current the *Sink* will draw during the *Explicit Contract*.

Note: A *Source* in *AVS Mode*, unlike the *SPR Source* in *PPS Mode*, does not support current limit; the *Sink* is responsible not to take more current than it requested.

A new **Request** / **EPR_Request** Message, with an updated **Operating Current** value, **Shall** be issued whenever the *Sink's* power needs change.

The value in the **Operating Current** field **Shall Not** exceed the value in the *Maximum Current* field of the **Source_Capabilities** Message. For *EPR AVS*, the **Operating Current** field **Shall Not** exceed the **PDP** / Output voltage rounded down to the nearest 50 mA.

This field **Shall** apply to the *Fixed Supply*, *Variable Supply*, *Programmable* and *AVS RDOs*.

6.4.2.9 Maximum Operating Current

The *Maximum Operating Current* field has been functionally **Deprecated**. In order to maintain backward compatibility with *Sources* that may try to interpret the **Maximum Operating Current** field in the **Request** Message or **EPR_Request** Message, the field **Shall** be set equal to the value of the **Operating Current** field. To ensure backward compatibility, the *Source* **Should Ignore** this field.

This field **Shall** apply to the *Fixed Supply* and *Variable Supply RDO* in *SPR Mode* and the *Fixed Supply RDO* in **EPR Capable**.

6.4.2.10 Operating Power

The **Operating Power** field in the *Request Data Object* **Shall** be set to the highest power the *Sink* will draw throughout the *Explicit Contract*.

This field **Shall** apply to the *Battery Supply RDO*.

6.4.2.11 Maximum Operating Power

The **Maximum Operating Power** field has been functionally **Deprecated**. In order to maintain backward compatibility with *Sources* that may try to interpret the **Maximum Operating Power** field in the *Request Message*, the field **Shall** be set equal to the value of the **Operating Power** field. To ensure backward compatibility, the *Source* **Should Ignore** this field.

This field **Shall** apply to the *Battery Supply RDO*.

6.4.2.12 Output Voltage

The Output Voltage field in the Programmable and *AVS Request Data Objects* **Shall** be set by the *Sink* to the voltage the *Sink* requires as measured at the *Source's* output connector. The Output Voltage field **Shall** be greater than or equal to the Minimum Voltage field and less than or equal to the Maximum Voltage field in the Programmable Power Supply and *AVS APDOs*, respectively.

This field **Shall** apply to the Programmable *RDO* and *AVS RDO*.

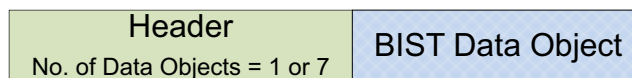
6.4.3 BIST Message

The **BIST Message** is sent to request the *Port* to enter a *PHY Layer* test mode (see [Section 5.9, "Built in Self-Test \(BIST\)"](#)) that performs one of the following functions:

- Enter a *Continuous BIST Mode* to send a continuous stream of test data to the *Tester*.
- Enter and leave a *Shared Capacity Group* test mode.

The *Message* format is as shown in [Figure 6.14, "BIST Message"](#).

Figure 6.14 BIST Message



All Ports **Shall** be able to be a *Unit Under Test (UUT)* only when operating at **vSafe5V**. All of the following *BIST Modes* **Shall** be supported:

- Process reception of a **BIST Carrier Mode** *BIST Data Object* that **Shall** result in the generation of the appropriate carrier signal.
- Process reception of a **BIST Test Data** *BIST Data Object* that **Shall** result in the *Message* being **Ignored**.

UUTs with Ports constituting a *Shared Capacity Group* (see [\[USB Type-C 2.4\]](#)) **Shall** support the following *BIST Mode*:

- Process reception of a **BIST Shared Test Mode Entry** *BIST Data Object* that **Shall** cause the *UUT* to enter *BIST Shared Capacity Test Mode*; a mode in which the *UUT* offers its full *Source Capabilities* on every *Port* in the *Shared Capacity Group*.

- Process reception of a **BIST Shared Test Mode Exit** BIST Data Object that **Shall** cause the UUT to exit the BIST Shared Capacity Test Mode.

When a Port receives a **BIST** Message BIST Data Object for a BIST Mode when not operating at **vSafe5V**, the **BIST** Message **Shall** be **Ignored**.

When a Port receives a **BIST** Message BIST Data Object for a BIST Mode it does not support the **BIST** Message **Shall** be **Ignored**.

When a Port or Cable Plug receives a **BIST** Message BIST Data Object for a Continuous BIST Mode the Port or Cable Plug enters the requested BIST Mode and **Shall** remain in that BIST Mode for **tBISTContMode** and then **Shall** return to normal operation (see [Section 6.6.7.2, "BISTContModeTimer"](#)).

The usage model of the PHY Layer BIST Modes generally assumes that some controlling agent will request a test of its Port Partner.

In [Section 8.3.2.15, "Built in Self-Test \(BIST\)"](#) there is a sequence description of the test sequences used for compliance testing.

The fields in the BIST Data Object are defined in the [Table 6.27, "BIST Data Object"](#).

Table 6.27 BIST Data Object

Bit(s)	Value	Parameter	Description	Reference	Applicability
B31...28	0000b...0100b	Reserved	Shall Not be used	Section 1.4.2	-
	0101b	BIST Carrier Mode	Request Transmitter to enter BIST Carrier Mode	Section 6.4.3.1	Mandatory
	0110b...0111b	Reserved	Shall Not be used	Section 1.4.2	-
	1000b	BIST Test Data	Sends a Test Frame.	Section 6.4.3.2	Mandatory
	1001b	BIST Shared Test Mode Entry	Requests UUT to enter BIST Shared Capacity Test Mode.	Section 6.4.3.3.1	Mandatory for UUTs with shared capacity
	1010b	BIST Shared Test Mode Exit	Requests UUT to exit BIST Shared Capacity Test Mode.	Section 6.4.3.3.2	Mandatory for UUTs with shared capacity
	1011b...1111b	Reserved	Shall Not be used	Section 1.4.2	-
B27...0		Reserved	Shall be set to zero.	Section 1.4.2	-

6.4.3.1 BIST Carrier Mode

Upon receipt of a **BIST** Message, with a **BIST Carrier Mode** BIST Data Object, the UUT **Shall** send out a continuous string of BMC encoded alternating "1"s and "0"s.

The UUT **Shall** exit the Continuous BIST Mode within **tBISTContMode** of this Continuous BIST Mode being enabled (see [Section 6.6.7.2, "BISTContModeTimer"](#)).

6.4.3.2 BIST Test Data Mode

Upon receipt of a **BIST** Message, with a **BIST Test Data** BIST Data Object, the UUT **Shall** return a **GoodCRC** Message and **Shall** enter BIST Test Data Mode in which it sends no further Messages except for **GoodCRC** Messages in response to received Messages. See [Section 5.9.2, "BIST Test Data Mode"](#) for the definition of the Test Frame.

The test **Shall** be ended by sending **Hard Reset Signaling** to reset the UUT.

6.4.3.3 BIST Shared Capacity Test Mode

A *Shared Capacity Group* of Ports share a common power source that is not capable of simultaneously powering all the ports to their full *Source Capabilities* (see [USB Type-C 2.4]). The *BIST Shared Capacity Test Mode* **Shall** only be implemented by ports in a *Shared Capacity Group*.

The *UUT Shared Capacity Group* of Ports **Shall** contain one or more Ports, designated as Master Ports, that recognize both the *BIST Shared Test Mode Entry* BIST Data Object and the *BIST Shared Test Mode Exit* BIST Data Object.

6.4.3.3.1 BIST Shared Test Mode Entry

When any master Port in a *Shared Capacity Group* receives a *BIST* Message with a *BIST Shared Test Mode Entry* BIST Data Object, while in the *PE_SRC_Ready* State, the UUT **Shall** enter a compliance test mode where the maximum *Source Capabilities* are always offered on every Port, regardless of the availability of shared power i.e., all shared power management is disabled.

Ports in the *Shared Capacity Group* that are not Master Ports **Shall Not** enter compliance mode on receiving the *BIST Shared Test Mode Entry* BIST Data Object.

Upon receipt of a *BIST* Message, with a *BIST Shared Test Mode Entry* BIST Data Object, the UUT **Shall** return a *GoodCRC* Message and **Shall** enter the *BIST Shared Capacity Test Mode*.

On entering this mode, the UUT **Shall** send a new *Source_Capabilities* Message from each Port in the *Shared Capacity Group* within *tBISTSharedTestMode*. The Tester will not exceed the shared capacity during this mode.

6.4.3.3.2 BIST Shared Test Mode Exit

Upon receipt of a *BIST* Message, with a *BIST Shared Test Mode Exit* BIST Data Object, the UUT **Shall** return a *GoodCRC* Message and **Shall** exit the *BIST Shared Capacity Test Mode*. If any other Message, aside from a *BIST* Message, with a *BIST Shared Test Mode Exit* BIST Data Object, is received while in *BIST Shared Capacity Test Mode* this **Shall Not** cause the UUT to exit the *BIST Shared Capacity Test Mode*.

On exiting the mode, the UUT **May** send a new *Source_Capabilities* Message to each Port in the *Shared Capacity Group* or the UUT **May** perform *ErrorRecovery* on each Port.

Ports in the *Shared Capacity Group* that are not Master Ports **Shall Not** exit compliance mode on receiving the *BIST Shared Test Mode Exit* BIST Data Object.

Ports in the *Shared Capacity Group* that are not Master Ports **Should Not** exit compliance mode on receiving the *BIST Shared Test Mode Exit* BIST Data Object.

- The UUT **Shall** exit *BIST Shared Capacity Test Mode* when It is powered off.
- The UUT **Shall** remain in *BIST Shared Capacity Test Mode* for any PD event (except when a *BIST Shared Test Mode Exit* BIST Data Object, is received); specifically the UUT **Shall** remain in *BIST Shared Capacity Test Mode* when any of the following PD events occurs:
 - *Hard Reset*
 - *Cable Reset*
 - *Soft Reset*
 - *Data Role Swap*
 - *Power Role Swap*
 - *Fast Role Swap*
 - *VCONN Swap*.
- The UUT **May** leave *BIST Shared Capacity Test Mode* if the Tester makes a request that exceeds the *Capabilities* of the UUT.

6.4.4 Vendor Defined Message

The **Vendor Defined Message** (VDM) is provided to allow vendors to exchange information outside of that defined by this specification.

A **Vendor Defined Message** **Shall** consist of at least one *Vendor Data Object* (VDO), the *VDM Header*, and **May** contain up to a maximum of six additional VDOs.

To ensure vendor uniqueness of **Vendor Defined Messages**, all **Vendor Defined Messages** **Shall** contain a **Valid USB Standard or Vendor ID** (SVID) allocated by USB-IF in the *VDM Header*.

Two types of **Vendor Defined Messages** are defined: *Structured VDMs* and *Unstructured VDMs*. A *Structured VDM* defines an extensible structure designed to support *Modal Operation*. An *Unstructured VDM* does not define any structure and *Messages* **May** be created in any manner that the vendor chooses.

Vendor Defined Messages **Shall Not** be used for direct power *Negotiation*. They **May** however be used to alter *Local Policy*, affecting what is offered or consumed via the normal PD *Messages*.

The *Message* format **Shall** be as shown in [Figure 6.15, "Vendor Defined Message"](#).

Figure 6.15 Vendor Defined Message



The *VDM Header* **Shall** be the first 4-byte object in a *Vendor Defined Message*. The *VDM Header* provides *Command* space to allow vendors to customize *Messages* for their own purposes. Additionally, vendors **May** make use of the *Commands* in a *Structured VDM*.

The fields in the *VDM Header* for an *Unstructured VDM*, when the **VDM Type** Bit is set to zero, **Shall** be as defined in [Table 6.28, "Unstructured VDM Header"](#). The fields in the *VDM Header* for a *Structured VDM*, when the **VDM Type** Bit is set to one **Shall** be as defined in [Table 6.29, "Structured VDM Header"](#).

Both *Unstructured VDMs* and *Structured VDMs* **Shall** only be sent and received after an *Explicit Contract* has been established. The only exception to this is the **Discover Identity** Command which **May** be sent by *Source* when a *Default Contract* or an *Implicit Contract* (in place after *Attach*, a *Power Role Swap* or *Fast Role Swap*) is in place in order to discover *Cable Capabilities* (see [Section 8.3.3.25.3, "Source Startup Structured VDM Discover Identity of a Cable Plug State Diagram"](#)).

6.4.4.1 Unstructured VDM

The *Unstructured VDM* does not define the contents of bits B14...0 in the *VDM Header*. Their definition and use are the sole responsibility of the vendor indicated by the *VID*. The *Port Partners* and *Cable Plugs* **Shall** exit any states entered using an *Unstructured VDM* when a *Hard Reset* appears on PD.

The following rules apply to the use of *Unstructured VDM Messages*:

- *Unstructured VDMs* **Shall** only be used when an *Explicit Contract* is in place.
- Prior to establishing an *Explicit Contract* *Unstructured VDMs* **Shall Not** be sent and **Shall** be **Ignored** if received.
- Only the *DFP* **Shall** be an *Initiator* of *Unstructured VDMs*.
- Only the *UFP* or a *Cable Plug* **Shall** be a *Responder* to *Unstructured VDM*.
- *Unstructured VDMs* **Shall Not** be initiated or responded to under any other circumstances.

- *Unstructured VDMs* **Shall** only be used during *Modal Operation* in the context of an *Active Mode* i.e., only after the *UFP* has Ack'ed the *Enter Mode Command* can *Unstructured VDMs* be sent or received. The *Active Mode* and the associated *Unstructured VDMs* **Shall** use the same *SVID*.
- *Unstructured VDMs* **May** be used with *SOP* Packets*.
- When a *DFP* or *UFP* does not support *Unstructured VDMs* or does not recognize the *VID* it **Shall** return a *Not_Supported Message*.

[Table 6.28, "Unstructured VDM Header"](#) illustrates the *VDM Header* bits.

Table 6.28 Unstructured VDM Header

Bit(s)	Parameter	Description
B31...16	<i>Vendor ID (VID)</i>	Unique 16-bit unsigned integer. Assigned by the USB-IF to the Vendor.
B15	<i>VDM Type</i>	0 = <i>Unstructured VDM</i>
B14...0	Available for Vendor Use	Content of this field is defined by the vendor.

6.4.4.1.1 USB Vendor ID

The *Vendor ID (VID)* field **Shall** contain the 16-bit *Vendor ID* value assigned to the vendor by the USB-IF (*VID*). No other value **Shall** be present in this field.

6.4.4.1.2 VDM Type

The *VDM Type* field **Shall** be set to zero indicating that this is an *Unstructured VDM*.

6.4.4.2 Structured VDM

Setting the *VDM Type* field to 1 (*Structured VDM*) defines the use of bits B14...0 in the *Structured VDM Header*. The fields in the *Structured VDM Header* are defined in [Table 6.29, "Structured VDM Header"](#).

The following rules apply to the use of *Structured VDM Messages*:

- *Structured VDMs* **Shall** only be used when an *Explicit Contract* is in place with the following exception:
 - Prior to establishing the *First Explicit Contract*, a *Source* **May** issue *Discover Identity Messages*, to a *Cable Plug* using *SOP' Packets*, as an *Initiator* (see [Section 8.3.3.25.3, "Source Startup Structured VDM Discover Identity of a Cable Plug State Diagram"](#)).
- Either *Port* **May** be an *Initiator* of *Structured VDMs* except for the *Enter Mode* and *Exit Mode Commands* which **Shall** only be initiated by the *DFP*.
- A *Cable Plug* **Shall** only be a *Responder* to *Structured VDMs*.
- *Structured VDMs* **Shall Not** be initiated or responded to under any other circumstances.
- When a *DFP* or *UFP* does not support *Structured VDMs* any *Structured VDMs* received **Shall** return a *Not_Supported Message*.
- When using any of the *SVID Specific Commands* in the *Structured VDM Header* (*VDM Header* b4...0 - value 16 - 31) the *Responder* **Shall NAK** Messages where the *SVID* in the *VDM Header* is not recognized as an *SVID* that uses *SVID Specific Commands* or the use of *SVID Specific Commands* is not supported for the *SVID*.
- When a *Cable Plug* does not support *Structured VDMs* any *Structured VDMs* received **Shall** be *Ignored*.

A DFP, UFP or Cable Plug which supports *Structured VDMs* and receiving a *Structured VDM* for a *SVID* that it does not recognize **Shall** reply with a **NAK** Command.

Table 6.29 Structured VDM Header

Bit(s)	Field	Description
B31...16	Standard or Vendor ID (SVID)	Unique 16-bit unsigned integer, assigned by the USB-IF
B15	VDM Type	1 = <i>Structured VDM</i>
B14...13	Structured VDM Version (Major)	Version Number (Major) of the <i>Structured VDM</i> (not this specification Version): <ul style="list-style-type: none">Version 1.0 = 00b (Deprecated and Shall Not be used)Version 2.x = 01bValues 2-3 are Reserved and Shall Not be used
B12...11	Structured VDM Version (Minor)	For <i>Commands</i> 0...15 Version Number (Minor) of the <i>Structured VDM</i> <ul style="list-style-type: none">Version 2.0 = 00b (Used for ports implemented prior to USB PD Revision 3.1, Version 1.6)Version 2.1 = 01b (Used for ports implemented starting with USB PD Revision 3.1, Version 1.6)All other Values are Reserved and Shall Not be used<i>SVID Specific Commands</i> (16...31) defined by the <i>SVID</i>.
B10...8	Object Position	For the Enter Mode , Exit Mode , and Attention <i>Commands</i> (Requests/Responses): <ul style="list-style-type: none">000b = Reserved and Shall Not be used.001b...110b = Index into the list of <i>VDOs</i> to identify the desired <i>Alternate Mode VDO</i>111b = Exit all <i>Active Modes</i> (equivalent of a power on reset). Shall only be used with the <i>Command</i>. <i>Commands</i> 0...3, 7...15: <ul style="list-style-type: none">000b001b...111b = Reserved and Shall Not be used. <i>SVID Specific Commands</i> (16...31) defined by the <i>SVID</i> .
B7...6	Command Type	00b = REQ (Request from <i>Initiator Port</i>)
		01b = ACK (Acknowledge Response from <i>Responder Port</i>)
		10b = NAK (Negative Acknowledge Response from <i>Responder Port</i>)
		11b = BUSY (Busy Response from <i>Responder Port</i>)
B5	Reserved	Shall be set to zero and Shall be Ignored
B4...0 ¹	Command	0 = Reserved and Shall Not be used.
		1 = Discover Identity
		2 = Discover SVIDs
		3 = Discover Modes
		4 = Enter Mode
		5 = Exit Mode
		6 = Attention
		7-15 = Reserved and Shall Not be used.
16...31 = <i>SVID Specific Commands</i>		
1) In the case where a SID is used the modes are defined by a standard. When a VID is used the modes are defined by the Vendor.		

[Section Table 6.30, "Structured VDM Commands"](#) shows the *Commands*, which *SVID* to use with each *Command* and the *SOP** values which **Shall** be used.

Table 6.30 Structured VDM Commands

Command	VDM Header SVID Field	SOP* used
<i>Discover Identity</i>	Shall only use the PD SID .	Shall only use <i>SOP/SOP'</i> .
<i>Discover SVIDs</i>	Shall only use the PD SID .	Shall only use <i>SOP/SOP'</i> .
<i>Discover Modes</i>	Valid with any <i>SVID</i> .	Shall only use <i>SOP/SOP'</i> .
<i>Enter Mode</i>	Valid with any <i>SVID</i> .	Valid with <i>SOP*</i> .
<i>Exit Mode</i>	Valid with any <i>SVID</i> .	Valid with <i>SOP*</i> .
<i>Attention</i>	Valid with any <i>SVID</i> .	Valid with <i>SOP*</i> .
<i>SVID Specific Commands</i>	Valid with any <i>SVID</i> .	Valid with <i>SOP*</i> (defined by <i>SVID</i>).

6.4.4.2.1 SVID

The **Standard or Vendor ID (SVID)** field **Shall** contain either a 16-bit USB Standard ID value (SID) or the 16-bit assigned to the vendor by the USB-IF (*VID*). No other value **Shall** be present in this field.

[Section Table 6.31, "SVID Values"](#) lists specific *SVID* values referenced by this specification.

Table 6.31 SVID Values

Parameter	Value	Description
PD SID	0xFF00	Standard ID allocated to this specification by USB-IF.
DPTC SID	0xFF01	Standard ID allocated to [DPTC2.1] by USB-IF.

6.4.4.2.2 VDM Type

The **VDM Type** field **Shall** be set to one indicating that this is a *Structured VDM*.

6.4.4.2.3 Structured VDM Version

The **Structured VDM Version (Major)/Structured VDM Version (Minor)** fields indicate the level of functionality supported in the *Structured VDM* part of the specification. This is not the same *Version* as the *Version* of this specification. The **Structured VDM Version (Major)** **Shall** be set to 01b to indicate Version 2.x with the **Structured VDM Version (Minor)** field set as appropriate based on whether the *Port* is implemented to USB PD Revision 3.1, Version 1.6 (or newer) or a prior *Version*.

To ensure interoperability with existing *PDUSB* products, *PDUSB* products **Shall** support every *Structured VDM Version* number starting from Version 1.0.

On receipt of a *VDM Header* with a higher *Version* number than it supports, a *Port* or *Cable Plug* **Shall** respond using the highest *Version* number it supports. On receipt of a *VDM Header* with a lower *Version* number than it supports, a *Port* or *Cable Plug* **Shall** respond using the same *Version* number it received.

The **Structured VDM Version (Major)/Structured VDM Version (Minor)** fields of the *Discover Identity* Command sent and received during the *Discovery Process* **Shall** be used to determine the lowest common *Structured VDM Version* supported by the *Port Partners* or *Cable Plug* and **Shall** continue to operate using this *Specification Revision* until they are *Detached*. After discovering the *Structured VDM Version*, the **Structured VDM Version (Major)/Structured VDM Version (Minor)** fields **Shall** match the agreed common *Structured VDM Version*.

6.4.4.2.4 Object Position

The **Object Position** field **Shall** be used by the **Enter Mode** and **Exit Mode** Commands. The **Discover Modes Command** returns a list of zero to six VDOs, each of which describes an *Alternate Mode*. The value in **Object Position** field is an index into that list that indicates which VDO (e.g., *Alternate Mode*) in the list the **Enter Mode** and **Exit Mode** Command refers to. The **Object Position** **Shall** start with one for the first *Alternate Mode* in the list. If the SVID is a VID, the content of the VDO for the *Alternate Mode* **Shall** be defined by the vendor. If the **Standard or Vendor ID (SVID)** is a SID, the value **Shall** be assigned, by the USB-IF, to the given Standard. The VDO's content **May** be as simple as a numeric value or as complex as bit mapped description of *Capabilities* of the *Alternate Mode*. In all cases, the *Responder* is responsible for deciphering the contents to know whether or not it supports the *Alternate Mode* at the Object Position.

This field **Shall** be set to zero in the Request or Response (**REQ**, **ACK**, **NAK** or **BUSY**) when not required by the specification of the individual *Command*.

6.4.4.2.5 Command Type

6.4.4.2.5.1 Commands other than Attention

This **Command Type** field **Shall** be used to indicate the type of *Command* request/response being sent.

An *Initiator* **Shall** set the **Command Type** field to **REQ** to indicate that this is a *Command* request from an *Initiator*.

If *Structured VDMs* are supported, then the responses are as follows:

- "Responder **ACK**" is the normal return and **Shall** be sent to indicate that the *Command* request was received and handled normally.
- "Responder **NAK**" **Shall** be returned when the *Command* request:
 - Has an **Invalid** parameter (e.g., **Invalid SVID** or *Alternate Mode*).
 - Cannot be acted upon because the configuration is not correct (e.g., an *Alternate Mode* which has a dependency on another *Alternate Mode* or a request to exit an *Alternate Mode* which is not an *Active Mode*).
 - Is an Unrecognized *Message*.
 - The handling of "Responder **NAK**" is left up to the *Initiator*.
- "Responder **BUSY**" **Shall** be sent in the response to a VDM when the *Responder* is unable to respond to the *Command* request immediately, but the *Command* request **May** be retried. The *Initiator* **Shall** wait **tVDMBusy** after a "Responder **BUSY**" response is received before retrying the *Command* request.

6.4.4.2.5.2 Attention Command

This **Command Type** field **Shall** be used to indicate the type of *Command* request being sent. An *Initiator* **Shall** set the field to **REQ** to indicate that this is a *Command* request from an *Initiator*. If *Structured VDMs* are supported, then no response **Shall** be made to an **Attention Command**.

6.4.4.2.6 Command

6.4.4.2.6.1 Commands other than Attention

The **Command** field contains the value for the VDM *Command* being sent. The *Commands* explicitly listed in the **Command** field are used to identify devices and manage their operational Modes. There is a further range of *Command* values left for the vendor to use to manage additional extensions.

A *Structured VDM Command* consists of a *Command* request and a *Command* response (**ACK**, **NAK** or **BUSY**). A *Structured VDM Command* is deemed to be completed (and if applicable, the transition to the requested functionality is made) when the **GoodCRC Message** has been successfully received by the *Responder* in reply to its *Command* response.

If *Structured VDMs* are supported, but the *Structured VDM Command* request is an *Unrecognized Message*, it **Shall** be *NAKed* (see [Table 6.32, "Commands and Responses"](#)).

6.4.4.2.6.2 Attention Command

The **Command** field contains the value for the *VDM Command* being sent (**Attention**). The **Attention Command** **May** be used by the *Initiator* to notify the *Responder* that it requires service.

A *Structured VDM Attention Command* consists of a *Command* request but no *Command* response. A *Structured VDM Attention Command* is deemed to be completed when the **GoodCRC** *Message* has been successfully received by the *Initiator* in reply to its **Attention Command** request.

If *Structured VDMs* are supported, but the *Structured VDM Attention Command* request is an *Unrecognized Message* it **Shall** be **Ignored** (see [Table 6.32, "Commands and Responses"](#)).

6.4.4.3 Use of Commands

The *VDM Header* for a *Structured VDM Message* defines *Commands* used to retrieve a list of *SVIDs* the device supports, to discover the *Modes* associated with each *SVID*, and to enter/exit the *Modes*. The *Commands* include:

- **Discover Identity**
- **Discover SVIDs**
- **Discover Modes**
- **Enter Mode**
- **Exit Mode**
- **Attention**

Additional *Command* space is also **Reserved** for Standard and Vendor use and for future extensions.

The *Command AMs* use the terms *Initiator* and *Responder* to identify messaging roles the ports are taking on relative to each other. This role is independent of the *Port's* power capability (*Provider*, *Consumer* etc.) or its present *Power Role* (*Source* or *Sink*). The *Initiator* is the *Port* sending the initial *Command* request and the *Responder* is the *Port* replying with the *Command* response. See [Section 6.4.4.4, "Command Processes"](#).

All *Ports* that support *Modes* **Shall** support the **Discover Identity**, **Discover SVIDs**, the **Discover Modes**, the **Enter Mode** and **Exit Mode** *Commands*.

[Table 6.32, "Commands and Responses"](#) details the responses a *Responder* **May** issue to each *Command* request. Responses not listed for a given *Command* **Shall Not** be sent by a *Responder*. A **NAK** response **Should** be taken as an indication not to retry that particular *Command*.

Table 6.32 Commands and Responses

Command	Allowed Response	Reference
Discover Identity	ACK, NAK, BUSY	Section 6.4.4.3.1
Discover SVIDs	ACK, NAK, BUSY	Section 6.4.4.3.2
Discover Modes	ACK, NAK, BUSY	Section 6.4.4.3.3
Enter Mode	ACK, NAK	Section 6.4.4.3.4
Exit Mode	ACK, NAK	Section 6.4.4.3.5
Attention	None	Section 6.4.4.3.6

Examples of *Command* usage can be found in [Appendix C, "VDM Command Examples"](#).

6.4.4.3.1 Discover Identity

The **Discover Identity** Command is provided to enable an *Initiator* to identify its *Port Partner* and for an *Initiator* (*VCONN Source*) to identify the *Responder* (*Cable Plug* or *VPD*). The **Discover Identity** Command is also used to determine whether a *Cable Plug* or *VPD* is PD-Capable by looking for a **GoodCRC** Message Response.

The **Discover Identity** Command **Shall** only be sent to *SOP* when there is an *Explicit Contract*.

The **Discover Identity** Command **Shall** be used to determine whether a given *Cable Plug* or *VPD* is PD Capable (see [Section 8.3.3.21.1, "Initiator Structured VDM Discover Identity State Diagram"](#) and [Section 8.3.3.25.3, "Source Startup Structured VDM Discover Identity of a Cable Plug State Diagram"](#)). In this case a **Discover Identity** Command request sent to *SOP* **Shall Not** cause a *Soft Reset* if a **GoodCRC** Message response is not returned since this can indicate a non-PD Capable cable or *VPD*.

Note: A *Cable Plug* or *VPD* will not be ready for PD Communication until *tVCONNStable* after *VCONN* has been applied (see [\[USB Type-C 2.4\]](#)).

During *Cable Plug* or *VPD* discovery, when there is an *Explicit Contract*, **Discover Identity** Commands are sent at a rate defined by the **DiscoverIdentityTimer** (see [Section 6.6.15, "DiscoverIdentityTimer"](#)) up to a maximum of **nDiscoverIdentityCount** times (see [Section 6.7.5, "Discover Identity Counter"](#)).

A PD-Capable *Cable Plug* or *VPD* **Shall** return a **Discover Identity** Command **ACK** in response to a **Discover Identity** Command request sent to *SOP*.

The **Discover Identity** Command **Shall** be used to determine the identity and/or *Capabilities* of the *Port Partner*. The following products **Shall** return a **Discover Identity** Command **ACK** in response to a **Discover Identity** Command request sent to *SOP*:

- A PD-Capable *UFP* that supports *Modal Operation*.
- A PD-Capable product that has multiple *DFPs*.
- A PD-Capable [\[USB4\]](#) product.

The *SVID* in the **Discover Identity** Command request **Shall** be set to the PD SID (see [Section Table 6.31, "SVID Values"](#)).

The **Number of Data Objects** field in the *Message Header* in the **Discover Identity** Command request **Shall** be set to 1 since the **Discover Identity** Command request **Shall Not** contain any *VDOs*.

The **Discover Identity** Command **ACK** sent back by the *Responder* **Shall** contain an *ID Header VDO*, a *Cert Stat VDO*, a *Product VDO* and the *Product Type VDOs* defined by the *Product Type* as shown in [Figure 6.16, "Discover Identity Command response"](#). This specification defines the following *Product Type VDOs*:

- *Passive Cable VDO* (see [Section 6.4.4.3.1.6, "Passive Cable VDO"](#))
- *Active Cable VDOs* (see [Section 6.4.4.3.1.7, "Active Cable VDOs"](#))
- *VCONN Powered USB Device (VPD) VDO* (see [Section 6.4.4.3.1.9, "VCONN Powered USB Device VDO"](#))
- *UFP VDO* (see [Section 6.4.4.3.1.4, "UFP VDO"](#))
- *DFP VDO* (see [Section 6.4.4.3.1.5, "DFP VDO"](#))

No *VDOs* other than those defined in this specification **Shall** be sent as part of the **Discover Identity** Command response. Where there is no *Product Type VDO* defined for a specific *Product Type*, no *VDOs* **Shall** be sent as part of the **Discover Identity** Command response. Any additional *VDOs* received by the *Initiator* **Shall** be **Ignored**.

Figure 6.16 Discover Identity Command response

Header No. of Data Objects = 4-7 ¹	VDM Header	ID Header VDO	Cert Stat VDO	Product VDO	0..3 ² Product Type VDO(s)
--	------------	---------------	---------------	-------------	---------------------------------------

1. Only Data objects defined in this specification can be sent as part of the *Discover Identity* Command.
2. The following sections define the number and content of the VDOs for each Product Type.

The **Number of Data Objects** field in the *Message Header* in the *Discover Identity* Command *NAK* and *BUSY* responses **Shall** be set to 1 since they **Shall Not** contain any VDOs.

If the product is a DRD both a *Product Type (UFP)* and a *Product Type (DFP)* are declared in the ID Header. These products **Shall** return *Product Type VDOs* for both *UFP* and *DFP* beginning with the *UFP VDO*, then by a 32-bit Pad Object (defined as all '0's), followed by the *DFP VDO* as shown in [Figure 6.17, "Discover Identity Command response for a DRD"](#).

Figure 6.17 Discover Identity Command response for a DRD

Header No. of Data Objects = 7	VDM Header	ID Header VDO	Cert Stat VDO	Product VDO	Product Type VDO(s)		
					UFP	Pad	DFP

6.4.4.3.1.1 ID Header VDO

The *ID Header VDO* contains information corresponding to the Power Delivery Product. The fields in the *ID Header VDO* **Shall** be as defined in [Section Table 6.33, "ID Header VDO"](#).

Table 6.33 ID Header VDO

Bit(s)	Description	Reference
B31	USB Communications Capable as USB Host <ul style="list-style-type: none"> Shall be set to one if the product is capable of enumerating <i>USB Devices</i>. Shall be set to zero otherwise. 	Section 6.4.4.3.1.1.1
B30	USB Communications Capable as a USB Device <ul style="list-style-type: none"> Shall be set to one if the product is capable of being enumerated as a <i>USB Device</i>. Shall be set to zero otherwise 	Section 6.4.4.3.1.1.2
B29...27	SOP Product Type (UFP) <ul style="list-style-type: none"> 000b – Not a <i>UFP</i> 001b – <i>PDUSB Hub</i> 010b – <i>PDUSB Peripheral</i> 011b – <i>PSD</i> 100b...111b – Reserved, Shall Not be used. 	Section 6.4.4.3.1.1.3
	SOP' Product Type (Cable Plug/VPD) <ul style="list-style-type: none"> 000b – Not a <i>Cable Plug/VPD</i> 001b...010b – Reserved, Shall Not be used. 011b – <i>Passive Cable</i> 100b – <i>Active Cable</i> 101b – Reserved, Shall Not be used. 110b – <i>VCONN Powered USB Device (VPD)</i> 111b – Reserved, Shall Not be used. 	

Table 6.33 ID Header VDO (Continued)

Bit(s)	Description	Reference
B26	Modal Operation Supported	Section 6.4.4.3.1.1.4
	<ul style="list-style-type: none"> Shall be set to one if the product (<i>UFP/Cable Plug</i>) is capable of supporting <i>Modal Operation (Alternate Modes)</i>. Shall be set to zero otherwise. 	
B25...23	SOP - Product Type (DFP) <ul style="list-style-type: none"> 000b – Not a <i>DFP</i> 001b – <i>PDUSB Hub</i> 010b – <i>PDUSB Host</i> 011b – Power Brick 100b...111b – Reserved, Shall Not be used. <i>SOP</i> : Reserved, Shall Not be used.	Section 6.4.4.3.1.1.6
B22...21	Connector Type <ul style="list-style-type: none"> 00b – Reserved, for compatibility with legacy systems. 01b – Reserved, Shall Not be used. 10b – <i>USB Type-C Receptacle</i> 11b – <i>USB Type-C Plug</i> 	Section 6.4.4.3.1.1.7
B20...16	Reserved, Shall Not be used.	
B15...0	USB Vendor ID	Section 6.4.4.3.1.1.8 [USB 2.0] / [USB 3.2] / [USB4]

6.4.4.3.1.1.1 USB Communications Capable as a USB Host

The **USB Communications Capable as USB Host** field is used to indicate whether or not the *Port* has a *USB Host* Capability.

6.4.4.3.1.1.2 USB Communications Capable as a USB Device

The **USB Communications Capable as a USB Device** field is used to indicate whether or not the *Port* has a *USB Device* Capability.

6.4.4.3.1.1.3 Product Type (UFP)

The **SOP Product Type (UFP)** field indicates the type of Product when in *UFP Data Role*, whether a *VDO* will be returned and if so the type of *VDO* to be returned. The *Product Type* indicated in the **SOP Product Type (UFP)** field **Shall** be the closest categorization of the main functionality of the Product in *UFP Data Role* or "Undefined" when there is no suitable category for the product. For DRD Products this field **Shall** always indicate the *Product Type* when in *UFP* role regardless of the present *Data Role*. [Table 6.34, "Product Types \(UFP\)"](#) defines the *Product Type VDOs* which **Shall** be returned.

Table 6.34 Product Types (UFP)

Product Type	Description	Product Type VDO	Reference
Undefined	Shall be used when this is not a <i>UFP</i> .	None	
<i>PDUSB Hub</i>	Shall be used when the Product is a <i>PDUSB Hub</i> .	<i>UFP VDO</i>	Section 6.4.4.3.1.1.4
<i>PDUSB Peripheral</i>	Shall be used when the Product is a <i>PDUSB Device</i> other than a <i>PDUSB Hub</i> .	<i>UFP VDO</i>	Section 6.4.4.3.1.1.4
PSD	Shall be used when the Product is a PSD, e.g., power bank.	None	

6.4.4.3.1.1.4 Product Type (Cable Plug)

The *SOP' Product Type (Cable Plug/VPD)* field indicates the type of Product when the Product is a *Cable Plug* or *VPD*, whether a *VDO* will be returned and if so the type of *VDO* to be returned. [Table 6.35, "Product Types \(Cable Plug/VPD\)"](#) defines the *Product Type VDOs* which **Shall** be returned.

Table 6.35 Product Types (Cable Plug/VPD)

Product Type	Description	Product Type VDO	Reference
Undefined	Shall be used where no other <i>Product Type</i> value is appropriate.	None	
Active Cable	Shall be used when the Product is a cable that incorporates signal conditioning circuits.	Active Cable VDO	Section 6.4.4.3.1.7
Passive Cable	Shall be used when the Product is a cable that does not incorporate signal conditioning circuits.	Passive Cable VDO	Section 6.4.4.3.1.6
VCONN Powered USB Device	Shall be used when the Product is a PDUSB VCONN Powered USB Device.	VPD VDO	Section 6.4.4.3.1.9

6.4.4.3.1.1.5 Modal Operation Supported

The **Modal Operation Supported** bit is used to indicate whether or not the Product (either a *Cable Plug* or a device that can operate in the *UFP* role) is capable of supporting Modes. The **Modal Operation Supported** bit does not describe a *DFP's Alternate Mode Controller* functionality.

A product that supports *Modal Operation* **Shall** respond to the **Discover SVIDs** Command with a list of *SVIDs* for all of the Modes it is capable of supporting whether or not those Modes can currently be entered.

6.4.4.3.1.1.6 Product Type (DFP)

The **SOP - Product Type (DFP)** field indicates the type of Product when in *DFP Data Role*, whether a *VDO* will be returned and if so the type of *VDO* to be returned. The *Product Type* indicated in the **SOP - Product Type (DFP)** field **Shall** be the closest categorization of the main functionality of the Product in *DFP Data Role* or "Undefined" when there is no suitable category for the product. For DRD Products this field **Shall** always indicate the *Product Type* when in *DFP* role regardless of the present *Data Role*. [Table 6.36, "Product Types \(DFP\)"](#) defines the *Product Type VDOs* which **Shall** be returned.

In *SOP' Communication (Cable Plugs and VPDs)* this bit field is **Reserved** and **Shall** be set to zero.

Table 6.36 Product Types (DFP)

Product Type	Description	Product Type VDO	Reference
Undefined	Shall be used where no other <i>Product Type</i> value is appropriate.	None	
PDUSB Hub	Shall be used when the Product is a <i>PDUSB Hub</i> .	DFP VDO	Section 6.4.4.3.1.7
PDUSB Host	Shall be used when the Product is a <i>PDUSB Host</i> or a <i>PDUSB</i> host that supports one or more <i>Alternate Modes</i> as an <i>AMC</i> .	DFP VDO	Section 6.4.4.3.1.6
Charger	Shall be used when the Product is a <i>Charger</i> .	DFP VDO	Section 6.4.4.3.1.9

6.4.4.3.1.1.7 Connector Type Field

The **Connector Type** field (B22...21) **Shall** contain a value identifying it as either a *USB Type-C* receptacle or a *USB Type-C* plug.

6.4.4.3.1.1.8

Vendor ID

Manufacturers **Shall** set the **USB Vendor ID** field to the value of the *Vendor ID* assigned to them by USB-IF. For *USB Devices* or *Hubs* which support *USB Communications* the **USB Vendor ID** field **Shall** be identical to the Vendor ID field defined in the product's *USB Device Descriptor* (see [\[USB 2.0\]](#) and [\[USB 3.2\]](#)).

6.4.4.3.1.2

Cert Stat VDO

The *Cert Stat VDO* **Shall** contain the XID assigned by USB-IF to the product before certification in binary format. The fields in the *Cert Stat VDO* **Shall** be as defined in [Table 6.37, "Cert Stat VDO"](#).

Table 6.37 Cert Stat VDO

Bit(s)	Description	Reference
B31...0	32-bit unsigned integer, XID	Assigned by USB-IF

6.4.4.3.1.3

Product VDO

The *Product VDO* contains identity information relating to the product. The fields in the *Product VDO* **Shall** be as defined in [Table 6.38, "Product VDO"](#).

Table 6.38 Product VDO

Bit(s)	Description	Reference
B31...16	16-bit unsigned integer, USB Product ID	[USB 2.0] / [USB 3.2]
B15...0	16-bit unsigned integer, bcdDevice	[USB 2.0] / [USB 3.2]

Manufacturers **Should** set the USB Product ID field to a unique value identifying the product and **Should** set the bcdDevice field to a version number relevant to the release version of the product.

6.4.4.3.1.4

UFP VDO

The *UFP VDO* defined in this section **Shall** be returned by Ports capable of operating as a *UFP* including traditional USB peripherals, *USB Hub*'s upstream *Port* and DRD capable host Ports. The *UFP VDO* defined in this section **Shall** be sent when the *Product Type* (*UFP*) field in the *ID Header VDO* is given as a *PDUSB Peripheral* or *PDUSB Hub*. [Table 6.39, "UFP VDO"](#) defines the *UFP VDO* that **Shall** be sent based on the *Product Type*.

A [\[USB4\]](#) *UFP* **Shall** support the *Structured VDM Discover Identity* Command.

Table 6.39 UFP VDO

Bit(s)	Description	Reference	
B31...29	UFP VDO Version	Version Number of the <i>VDO</i> (not this specification <i>Version</i>): <ul style="list-style-type: none">Version 1.3 = 011b Values 100b...111b are Reserved, Shall Not be used.	
B28	Reserved	Shall be set to zero.	
B27...24	Device Capability	Bit	Description
		0	[USB 2.0] Device Capable
		1	[USB 2.0] Device Capable (Billboard only)
		2	[USB 3.2] Device Capable
B23...22	Connector Type (Legacy)	3	[USB4] Device Capable
		Deprecated, Shall be set to 00b.	
B21...11	Reserved	Shall be set to zero.	

Table 6.39 UFP VDO (Continued)

Bit(s)	Description	Reference								
B10...8	<i>VCONN Power</i>	<p>When the <i>VCONN Required</i> field is set to “Yes” the <i>VCONN Power</i> Field indicates the <i>VCONN</i> power needed by the <i>AMA</i> for full functionality:</p> <ul style="list-style-type: none">• 000b = 1W• 001b = 1.5W• 010b = 2W• 011b = 3W• 100b = 4W• 101b = 5W• 110b = 6W <p>111b = Reserved, Shall Not be used.</p> <p>When the <i>VCONN Required</i> field is set to “No” the <i>VCONN Power</i> field is Reserved and Shall be set to zero.</p>								
B7	<i>VCONN Required</i>	<p>Indicates whether the <i>AMA</i> requires <i>VCONN</i> in order to function.</p> <ul style="list-style-type: none">• 0 = No• 1 = Yes <p>When the <i>Alternate Modes</i> field indicates no modes are supported, the <i>VCONN Required</i> field is Reserved and Shall be set to zero.</p>								
B6	<i>VBUS Required</i>	<p>Indicates whether the <i>AMA</i> requires <i>VBUS</i> in order to function.</p> <ul style="list-style-type: none">• 0 = Yes• 1 = No <p>When the <i>Alternate Modes</i> field indicates no modes are supported, the <i>VBUS Required</i> field is Reserved and Shall be set to zero.</p>								
B5...3	<i>Alternate Modes</i>	<table><tr><th>Bit</th><th>Description</th></tr><tr><td>0</td><td>Supports <i>[TBT3] Alternate Mode</i></td></tr><tr><td>1</td><td>Supports <i>Alternate Modes</i> that reconfigure the signals on the <i>[USB Type-C 2.4]</i> connector – except for <i>[TBT3]</i>.</td></tr><tr><td>2</td><td>Supports <i>Alternate Modes</i> that do not reconfigure the signals on the <i>[USB Type-C 2.4]</i> connector.</td></tr></table>	Bit	Description	0	Supports <i>[TBT3] Alternate Mode</i>	1	Supports <i>Alternate Modes</i> that reconfigure the signals on the <i>[USB Type-C 2.4]</i> connector – except for <i>[TBT3]</i> .	2	Supports <i>Alternate Modes</i> that do not reconfigure the signals on the <i>[USB Type-C 2.4]</i> connector.
Bit	Description									
0	Supports <i>[TBT3] Alternate Mode</i>									
1	Supports <i>Alternate Modes</i> that reconfigure the signals on the <i>[USB Type-C 2.4]</i> connector – except for <i>[TBT3]</i> .									
2	Supports <i>Alternate Modes</i> that do not reconfigure the signals on the <i>[USB Type-C 2.4]</i> connector.									
B2...0	<i>USB Highest Speed</i>	<ul style="list-style-type: none">• 000b = <i>[USB 2.0]</i> only, no SuperSpeed support• 001b = <i>[USB 3.2]</i> Gen1• 010b = <i>[USB 3.2]/[USB4]</i> Gen2• 011b = <i>[USB4]</i> Gen3• 100b = <i>[USB4]</i> Gen4• 101b...111b = Reserved and Shall be set to zero.								

6.4.4.3.1.4.1 VDO Version Field

The *UFP VDO Version* field contains a *VDO Version* for this *VDM Version* number. This field indicates the expected content for the *UFP VDOs*.

6.4.4.3.1.4.2 Device Capability Field

The *Device Capability* bit-field describes the *UFP's Capabilities* when operating as either a *PDUSB Device* or *PDUSB Hub*.

The bits in the bit-field **Shall** be non-zero when the corresponding *USB Device* speed is supported and **Shall** be set to zero when the corresponding *USB Device* speed is not supported.

[**USB 2.0**] "Device capable" and "Device capable Billboard only" (bits 0 and 1) **Shall Not** be simultaneously set.

6.4.4.3.1.4.3 Connector Type Field

The **Connector Type (Legacy)** field was previously used for the *UFP VDO*'s Connector Type. **Shall** be set to 00b by the *Cable Plug* and **Shall** be **Ignored** by the receiver. The receiver can find this information in the **Connector Type** field in the *ID Header VDO* ([Section 6.4.4.3.1.1.7, "Connector Type Field"](#)).

6.4.4.3.1.4.4 VCONN Power Field

When the **VCONN Required** field indicates that *VCONN* is required the **VCONN Power** field **Shall** indicate how much power an *AMA* needs in order to fully operate. When the **VCONN Required** field is set to "No" the **VCONN Power** field is **Reserved** and **Shall** be set to zero.

6.4.4.3.1.4.5 VCONN Required Field

The **VCONN Required** field **Shall** indicate whether *VCONN* is needed for the *AMA* to operate. The **VCONN Required** field **Shall** only be used if the **Alternate Modes** field indicates that an *Alternate Mode* is supported. If no *Alternate Modes* are supported, this field is **Reserved** and **Shall** be set to zero.

6.4.4.3.1.4.6 VBUS Required Field

The **VBUS Required** field **Shall** indicate whether *VBUS* is needed for the *AMA* to operate. The *VBUS* required field **Shall** only be used if the **Alternate Modes** field indicates that an *Alternate Mode* is supported. If no *Alternate Modes* are supported, this field is **Reserved** and **Shall** be set to zero.

6.4.4.3.1.4.7 Alternate Modes Field

The **Alternate Modes** field **Shall** be used to identify all the types of *Alternate Modes*, if any, a device supports.

6.4.4.3.1.4.8 USB Highest Speed Field

The **USB Highest Speed** field **Shall** indicate the *Port*'s highest speed capability. The *DFP* **Shall** consider all values indicated in this field that are higher than the highest value that the *DFP* recognizes as being **Valid** and functionally compatible with the highest speed that the *DFP* supports.

6.4.4.3.1.5 DFP VDO

The **DFP VDO** **shall** be returned by Ports capable of operating as a **DFP**; including those implemented by *Hosts, Hubs* and *Power Bricks*. The **DFP VDO** **shall** be returned when the *Product Type (DFP)* field in the *ID Header VDO* is given as *Power Brick, PDUSB Host* or *PDUSB Hub*. [Table 6.40, "DFP VDO"](#) defines the **DFP VDO** that **shall** be sent.

Table 6.40 DFP VDO

Bit(s)	Field	Description	
B31...29	DFP VDO Version	Version Number of the VDO (not this specification Version): <ul style="list-style-type: none">Version 1.2 = 010b Values 011b...111b are Reserved and shall Not be used	
B28...27	Reserved	shall be set to zero.	
B26...24	Host Capability	Bit	Description
		0	[USB 2.0] Host Capable
		1	[USB 3.2] Host Capable
		2	[USB4] Host Capable
B23...22	Connector Type (Legacy)	shall be set to 00b.	
B21...5	Reserved	shall be set to zero.	
B4...0	Port Number	Unique Port number to identify a specific Port on a multi-Port device.	

6.4.4.3.1.5.1 VDO Version Field

The **DFP VDO Version** field **shall** contain a **VDO Version** for this **VDM Version** number. This field indicates the expected content for the **DFP VDO**.

6.4.4.3.1.5.2 Host Capability Field

The **Host Capability** bit-field **shall** describe whether the **DFP** can operate as a **PDUSB Host** and the **DFP's Capabilities** when operating as a **PDUSB Host**.

Power Bricks and **PDUSB Hubs** **shall** set the **Host Capability** bits to zero.

6.4.4.3.1.5.3 Connector Type Field

The **Connector Type (Legacy)** field was previously used for the **UFP VDO's** Connector Type. **shall** be set to 00b by the *Cable Plug* and **shall** be **Ignored** by the receiver. The receiver can find this information in the **Connector Type** field in the *ID Header VDO* ([Section 6.4.4.3.1.1.7, "Connector Type Field"](#)).

6.4.4.3.1.5.4 Port Number Field

The **Port Number** field **shall** be a **Static** unique number that unambiguously identifies each **[USB Type-C 2.4]** **DFP**, including **DRPs**, on the device.

Note: This number is independent of the USB Port number.

6.4.4.3.1.6

Passive Cable VDO

The *Passive Cable VDO* defined in this section **Shall** be sent when the *Product Type* is given as *Passive Cable*. [Table 6.41, "Passive Cable VDO"](#) defines the *Cable VDO* which **Shall** be sent.

A *Passive Cable* has a USB Plug on each end at least one of which is a *Cable Plug* supporting *SOP' Communication*. A *Passive Cable* **Shall Not** incorporate data bus signal conditioning circuits and hence has no concept of Super Speed Directionality. A *Passive Cable* **Shall** include a *V_{BUS}* wire and **Shall** only respond to *SOP' Communication*. *Passive Cables* **Shall** support the Structured VDM *Discover Identity* Command and **Shall** return the *Passive Cable VDO* in a *Discover Identity* Command *ACK* as shown in [Table 6.41, "Passive Cable VDO"](#).

Table 6.41 Passive Cable VDO

Bit(s)	Field	Description
B31...28	<i>HW Version</i>	0000b...1111b assigned by the <i>VID</i> owner
B27...24	<i>Firmware Version</i>	0000b...1111b assigned by the <i>VID</i> owner
B23...21	<i>VDO Version</i>	Version Number of the <i>VDO</i> (not this specification <i>Version</i>): <ul style="list-style-type: none"> Version 1.0 = 000b Values 001b...111b are Reserved and Shall Not be used.
B20	Reserved	Shall be set to zero.
B19...18	<i>USB Type-C plug to USB Type-C/Captive (Passive Cable)</i>	<ul style="list-style-type: none"> 00b = Reserved and Shall Not be used 01b = Reserved and Shall Not be used 10b = <i>USB Type-C</i> 11b = <i>Captive</i>
B17	<i>EPR Capable (Passive Cable)</i>	<ul style="list-style-type: none"> 0b – Cable is not <i>EPR Capable</i> 1b = Cable is <i>EPR Capable</i>
B16...13	<i>Cable Latency (Passive Cable)</i>	<ul style="list-style-type: none"> 0000b – Reserved and Shall Not be used 0001b – <10ns (~1m) 0010b – 10ns to 20ns (~2m) 0011b – 20ns to 30ns (~3m) 0100b – 30ns to 40ns (~4m) 0101b – 40ns to 50ns (~5m) 0110b – 50ns to 60ns (~6m) 0111b – 60ns to 70ns (~7m) 1000b – > 70ns (>~7m) Note: 1001b1111b Reserved and Shall Not be used
B12...11	<i>Cable Termination Type (Passive Cable)</i>	<ul style="list-style-type: none"> 00b = <i>V_{CONN}</i> not required. <i>Cable Plugs</i> that only support <i>Discover Identity</i> Commands Shall set these bits to 00b. 01b = <i>V_{CONN}</i> required 10b...11b = Reserved and Shall Not be used
B10...9	<i>Maximum V_{BUS} Voltage (Passive Cable)</i>	Maximum Cable <i>V_{BUS}</i> Voltage ² : <ul style="list-style-type: none"> 00b – 20V 01b – 30V¹ (Deprecated) 10b – 40V¹ (Deprecated) 11b – 50V
B8...7	Reserved	Shall be set to zero.
1)	Values no longer allowed. When present the field Shall be interpreted as if it was 00b.	
2)	<i>EPR Sinks</i> with a captive cable Shall report 50V.	

Table 6.41 Passive Cable VDO (Continued)

Bit(s)	Field	Description
B6...5	<i>V_{BUS} Current Handling Capability (Passive Cable)</i>	<ul style="list-style-type: none"> 00b = Reserved and Shall Not be used 01b = 3A 10b = 5A 11b = Reserved and Shall Not be used
B4...3	Reserved	Shall be set to zero.
B2...0	<i>USB Highest Speed (Passive Cable)</i>	<ul style="list-style-type: none"> 000b = [USB 2.0] only, no SuperSpeed support 001b = [USB 3.2] Gen1 010b = [USB 3.2]/[USB4] Gen2 011b = [USB4] Gen3 100b = [USB4] Gen4 101b...111b = Reserved and Shall Not be used
1)	Values no longer allowed. When present the field Shall be interpreted as if it was 00b.	
2)	<i>EPR Sinks</i> with a captive cable Shall report 50V.	

6.4.4.3.1.6.1 HW Version Field

The *HW Version* (B31...28) contains a HW version assigned by the *VID* owner.

6.4.4.3.1.6.2 FW Version Field

The *Firmware Version* field (B27...24) contains a FW version assigned by the *VID* owner.

6.4.4.3.1.6.3 VDO Version Field

The *VDO Version* field (B23...20) contains a *VDO Version* for this *VDM Version* number. This field indicates the expected content for this *VDO*.

6.4.4.3.1.6.4 USB Type-C plug to USB Type-C/Captive Field

The *USB Type-C plug to USB Type-C/Captive (Passive Cable)* field (B19...18) **Shall** contain a value indicating whether the opposite end from the *USB Type-C* plug is another *USB Type-C* plug (i.e., a detachable Standard *USB Type-C* Cable Assembly) or is a Captive Cable Assembly.

6.4.4.3.1.6.5 EPR Mode Capable

The *EPR Capable (Passive Cable)* bit is a **Static** bit which **Shall** only be set when the cable is specifically designed for safe operation when carrying up to 48 volts at 5 amps.

6.4.4.3.1.6.6 Cable Latency Field

The *Cable Latency (Passive Cable)* field (B16...13) **Shall** contain a value corresponding to the signal latency through the cable which can be used as an approximation for its length.

6.4.4.3.1.6.7 Cable Termination Type Field

The *Cable Termination Type (Passive Cable)* field (B12...11) **Shall** contain a value indicating whether the *Passive Cable* needs *VCONN* only initially in order to support the *Discover Identity Command*, after which it can be removed, or the *Passive Cable* needs *VCONN* to be continuously applied in order to power some feature of the *Cable Plug*.

6.4.4.3.1.6.8 Maximum V_{BUS} Voltage Field

The *Maximum V_{BUS} Voltage (Passive Cable)* field (B10...9) **Shall** contain the maximum voltage that **Shall** be *Negotiated* using a *Fixed Supply* over the cable as part of an *Explicit Contract* where the maximum voltage that **Shall** be applied to the cable is *vSrcNew* max + *vSrcValid* max. For example, when the *Maximum V_{BUS} Voltage (Passive Cable)* field is 20V, a *Fixed Supply* of 20V can be *Negotiated* as part of an *Explicit Contract* where the absolute maximum voltage that can be applied to the cable is 21.55V. Similarly, when the *Maximum V_{BUS} Voltage (Passive*

Cable) field is 50V, a *Fixed Supply* of 48V can be *Negotiated* as part of an *Explicit Contract* where the absolute maximum voltage that can be applied to the cable is 50.9V. *Maximum Vbus Voltage (Passive Cable)* field values of 01b and 10b (formerly 30V and 40V) **shall** be treated if they were 00b (20V).

6.4.4.3.1.6.9 VBUS Current Handling Capability Field

The *Vbus Current Handling Capability (Passive Cable)* field (B6...5) **shall** indicate whether the cable is capable of carrying 3A or 5A.

6.4.4.3.1.6.10 USB Highest Speed Field

The *USB Highest Speed (Passive Cable)* field (B2...0) **shall** indicate the highest rate the cable supports. The *DFP* **shall** consider all values indicated in this field that are higher than the highest value that the *DFP* recognizes as being **Valid** and functionally compatible with the highest speed that the *DFP* supports.

6.4.4.3.1.7

Active Cable VDOs

An *Active Cable* has a USB Plug on each end at least one of which is a *Cable Plug* supporting *SOP' Communication*. An *Active Cable* **Shall** incorporate data bus signal conditioning circuits and **May** have a concept of Super Speed Directionality on its Super Speed wires. An *Active Cable* **May** include a *VBUS* wire.

An *Active Cable*:

- **Shall** respond to *SOP' Communication*.
- **May** respond to *SOP'' Communication*.
- **Shall** support the Structured VDM *Discover Identity* Command.
- In the *Discover Identity* Command **ACK**:
 - **Shall** set the *Product Type* in the *ID Header VDO* to *Active Cable*.
 - **Shall** return the *Active Cable VDOs* defined in [Table 6.42, "Active Cable VDO1"](#) and [Table 6.43, "Active Cable VDO2"](#).

Table 6.42 Active Cable VDO1

Bit(s)	Field	Description
B31...28	HW Version	0000b...1111b assigned by the <i>VID</i> owner
B27...24	Firmware Version	0000b...1111b assigned by the <i>VID</i> owner
B23...21	VDO Version	Version Number of the <i>VDO</i> (not this specification <i>Version</i>): <ul style="list-style-type: none"> • Version 1.0 = 000b Values 001b...111b are Reserved and Shall Not be used.
B20	Reserved	Shall be set to zero.
B19...18	USB Type-C plug to USB Type-C/Captive	<ul style="list-style-type: none"> • 00b = Reserved and Shall Not be used • 01b = Reserved and Shall Not be used • 10b = <i>USB Type-C</i> • 11b = <i>Captive</i>
B17	EPR Capable (Active Cable)	<ul style="list-style-type: none"> • 0b – Cable is not <i>EPR Capable</i> • 1b = Cable is <i>EPR Capable</i>
B16...13	Cable Latency	<ul style="list-style-type: none"> • 0000b – Reserved and Shall Not be used • 0001b – <10ns (~1m) • 0010b – 10ns to 20ns (~2m) • 0011b – 20ns to 30ns (~3m) • 0100b – 30ns to 40ns (~4m) • 0101b – 40ns to 50ns (~5m) • 0110b – 50ns to 60ns (~6m) • 0111b – 60ns to 70ns (~7m) • 1000b – 1000ns (~100m) • 1001b – 2000ns (~200m) • 1010b – 3000ns (~300m) • 1001b1111b Reserved and Shall Not be used Note: Includes latency of electronics in <i>Active Cable</i> .
B12...11	Cable Termination Type (Active Cable)	<ul style="list-style-type: none"> • 00b...01b = Reserved and Shall Not be used • 10b = One end Active, one end passive, <i>VCONN</i> required • 11b = Both ends Active, <i>VCONN</i> required
1)	Values no longer allowed. When present the field Shall be interpreted as if it was 00b.	
2)	<i>EPR Sinks</i> with a captive cable Shall report 50V.	

Table 6.42 Active Cable VDO1 (Continued)

Bit(s)	Field	Description
B10...9	<i>Maximum VBUS Voltage (Active Cable)</i>	Maximum Cable <i>VBUS</i> voltage ² : <ul style="list-style-type: none"> 00b – 20V 01b – 30V¹ (Deprecated) 10b – 40V¹ (Deprecated) 11b – 50V
B8...7	Reserved	Shall be set to zero.
B8	<i>SBU Supported</i>	<ul style="list-style-type: none"> 0 = SBU connections supported 1 = SBU connections are not supported
B7	<i>SBU Type</i>	When SBU Supported = 1 this bit Shall be Ignored When SBU Supported = 0: <ul style="list-style-type: none"> 0 = SBU is passive 1 = SBU is active
B6...5	<i>VBUS Current Handling Capability (Active Cable)</i>	When <i>VBUS Through Cable</i> is “No”, this field Shall be Ignored . When <i>VBUS Through Cable</i> is “Yes”: <ul style="list-style-type: none"> 00b = Reserved and Shall Not be used 01b = 3A 10b = 5A 11b = Reserved and Shall Not be used
B4	<i>VBUS Through Cable</i>	<ul style="list-style-type: none"> 0 = No 1 = Yes
B3	<i>SOP” Controller Present</i>	<ul style="list-style-type: none"> 0 = No SOP” controller present 1 = SOP” controller present
B2...0	<i>USB Highest Speed (Active Cable)</i>	<ul style="list-style-type: none"> 000b = <i>[USB 2.0]</i> only, no SuperSpeed support 001b = <i>[USB 3.2]</i> Gen1 010b = <i>[USB 3.2]/[USB4]</i> Gen2 011b = <i>[USB4]</i> Gen3 100b = <i>[USB4]</i> Gen4 101b...111b = Reserved and Shall Not be used
1) Values no longer allowed. When present the field Shall be interpreted as if it was 00b.		
2) <i>EPR Sinks</i> with a captive cable Shall report 50V.		

Table 6.43 Active Cable VDO2

Bit(s)	Field	Description
B31...24	<i>Maximum Operating Temperature</i>	The maximum internal operating temperature in °C. It might or might not reflect the plug’s skin temperature.
B23...16	<i>Shutdown Temperature</i>	The temperature, in °C, at which the cable will go into thermal shutdown so as not to exceed the allowable plug skin temperature.
B15	Reserved	Shall be set to zero.
B14...12	<i>U3/CLd Power</i>	<ul style="list-style-type: none"> 000b: >10mW 001b: 5-10mW 010b: 1-5mW 011b: 0.5-1mW 100b: 0.2-0.5mW 101b: 50-200μW 110b: <50μW 111b: Reserved and Shall Not be used

Table 6.43 Active Cable VDO2 (Continued)

Bit(s)	Field	Description
B11	<i>U3 to U0 transition mode</i>	<ul style="list-style-type: none"> 0b: U3 to U0 direct 1b: U3 to U0 through U3S
B10	<i>Physical connection</i>	<ul style="list-style-type: none"> 0b = Copper 1b = Optical
B9	<i>Active element</i>	<ul style="list-style-type: none"> 0b = Active Re-driver 1b = Active Re-timer
B8	<i>USB4 Supported</i>	<ul style="list-style-type: none"> 0b = <i>[USB4]</i> supported 1b = <i>[USB4]</i> not supported
B7...6	<i>USB 2.0 Hub Hops Consumed</i>	Number of <i>[USB 2.0]</i> 'hub hops' cable consumes. <i>Shall</i> be set to zero if USB 2.0 not supported.
B5	<i>USB 2.0 Supported</i>	<ul style="list-style-type: none"> 0b = <i>[USB 2.0]</i> supported 1b = <i>[USB 2.0]</i> not supported
B4	<i>USB 3.2 Supported</i>	<ul style="list-style-type: none"> 0b = <i>[USB 3.2]</i> SuperSpeed supported 1b = <i>[USB 3.2]</i> SuperSpeed not supported
B3	<i>USB Lanes Supported</i>	<ul style="list-style-type: none"> 0b = One lane 1b = Two lanes
B2	<i>Optically Isolated Active Cable</i>	<ul style="list-style-type: none"> 0b = No 1b = Yes
B1	<i>USB4 Asymmetric Mode Supported</i>	<ul style="list-style-type: none"> 0b = No 1b = Yes <i>Shall</i> be set to zero if asymmetry is not supported.
B0	<i>USB Gen</i>	<ul style="list-style-type: none"> 0b = Gen 1 1b = Gen 2 or higher Note: See VDO1 USB Highest Speed for details of Gen supported.

6.4.4.3.1.7.1 HW Version Field

The *HW Version* field (B31...28) contains a HW version assigned by the *VID* owner.

6.4.4.3.1.7.2 FW Version Field

The *Firmware Version* field (B27...24) contains a FW version assigned by the *VID* owner.

6.4.4.3.1.7.3 VDO Version Field

The *VDO Version* field (B23...20) contains a *VDO Version* for this *VDM Version* number. This field indicates the expected content for the *Active Cable VDOs*.

6.4.4.3.1.7.4 Connector Type Field

The *USB Type-C plug to USB Type-C/Captive* field (B19...18) *Shall* contain a value indicating whether the opposite end from the *USB Type-C* plug is another *USB Type-C* plug (i.e., a detachable Standard *USB Type-C* Cable Assembly) or is a Captive Cable Assembly.

6.4.4.3.1.7.5 EPR Mode Capable

The *EPR Capable (Active Cable)* is a *Static* bit which *Shall* only be set when the cable is specifically designed for safe operation when carrying up to 48 volts at 5 amps.

6.4.4.3.1.7.6 Cable Latency Field

The *Cable Latency* field (B16...13) *Shall* contain a value corresponding to the signal latency through the cable which can be used as an approximation for its length.

6.4.4.3.1.7.7 Cable Termination Type Field

The **Cable Termination Type (Active Cable)** field (B12...11) **shall** contain a value corresponding to whether the *Active Cable* has one or two *Cable Plugs* requiring power from *VCONN*.

6.4.4.3.1.7.8 Maximum VBUS Voltage Field

The **Maximum VBUS Voltage (Active Cable)** field (B10...9) **shall** contain the maximum voltage that **shall** be *Negotiated* as part of an *Explicit Contract* where the maximum voltage that **shall** be applied to the cable is *vSrcNew* max + *vSrcValid* max. When this field is set to 20V, the cable will safely carry a Programmable Power Supply *APDO* of 20V where the absolute maximum voltage that can be applied to the cable is 21.55V. Similarly, when the **Maximum VBUS Voltage (Active Cable)** field is 50V, a *Fixed Supply* of 48V can be *Negotiated* as part of an *Explicit Contract* where the absolute maximum voltage that can be applied to the cable is 50.9V. **Maximum VBUS Voltage (Active Cable)** field values of 01b and 10b (formerly 30V and 40V) **shall** be treated if they were 00b (20V).

6.4.4.3.1.7.9 SBU Supported Field

The **SBU Supported** field (B8) **shall** indicate whether the cable supports the SBUs in the cable.

6.4.4.3.1.7.10 SBU Type Field

The **SBU Type** field (B7) **shall** indicate whether the SBUs are passive or active (e.g., digital).

6.4.4.3.1.7.11 VBUS Current Handling Capability Field

The **VBUS Current Handling Capability (Active Cable)** field (B6...5) **shall** indicate whether the cable is capable of carrying 3A or 5A. The **VBUS Current Handling Capability (Active Cable)** field **shall** only be **Valid** when the **VBUS Current Handling Capability (Active Cable)** field indicates an end-to-end *VBUS* wire.

6.4.4.3.1.7.12 VBUS Through Cable Field

The **VBUS Through Cable** field (B4) **shall** indicate whether the cable contains an end-to-end *VBUS* wire.

6.4.4.3.1.7.13 SOP" Controller Present Field

The **SOP" Controller Present** field (B3) **shall** indicate whether one of the *Cable Plugs* is capable of *SOP"* Communication in addition to the **Normative SOP' Communication**.

6.4.4.3.1.7.14 USB Highest Speed Field

The **USB Highest Speed (Active Cable)** field (B2...0) **shall** indicate the highest rate the cable supports. The *DFP* **shall** consider all values indicated in this field that are higher than the highest value that the *DFP* recognizes as being **Valid** and functionally compatible with the highest speed that the *DFP* supports.

6.4.4.3.1.7.15 Maximum Operating Temperature Field

Maximum Operating Temperature field (B31...24) **shall** report the maximum allowable operating temperature inside the plug in °C.

6.4.4.3.1.7.16 Shutdown Temperature Field

Shutdown Temperature field (B23...16) **shall** indicate the temperature inside the plug, in °C, at which the plug will shut down its active signaling components. When this temperature is reached, it will be reported in the *Active Cable Status Message* through the Thermal Shutdown bit.

6.4.4.3.1.7.17 U3/CLd Power Field

The **U3/CLd Power** field (B14...12) **shall** indicate the power the cable consumes while in *[USB 3.2] U3* or *[USB4] CLd*.

6.4.4.3.1.7.18 U3 to U0 Transition Mode Field

The **U3 to U0 transition mode** field (B11) **shall** indicate which U3 to U0 mode the cable supports. This does not include the power in U3S if supported.

6.4.4.3.1.7.19 Physical Connection Field

The **Physical connection** field (B10) **Shall** indicate the cable's construction, whether the connection between the active elements is copper or optical.

6.4.4.3.1.7.20 Active element Field

The **Active element** field (B9) **Shall** indicate the cable's active element, whether the active element is a re-timer or a re-driver.

6.4.4.3.1.7.21 USB4 Supported Field

The **USB4 Supported** field (B8) **Shall** indicate whether or not the cable supports **[USB4]** operation.

6.4.4.3.1.7.22 USB 2.0 Hub Hops Consumed field

The **USB 2.0 Hub Hops Consumed** field (B7...6) **Shall** indicate the number of USB 2.0 'hub hops' that are lost due to the transmission time of the cable.

6.4.4.3.1.7.23 USB 2.0 Supported Field

The **USB 2.0 Supported** field (B5) **Shall** indicate whether or not the cable supports **[USB 2.0]** only signaling.

6.4.4.3.1.7.24 USB 3.2 Supported Field

The **USB 3.2 Supported** field (B4) **Shall**, indicate whether or not the cable supports **[USB 3.2]** SuperSpeed signaling.

6.4.4.3.1.7.25 USB Lanes Supported Field

The **USB Lanes Supported** field (B3) **Shall** indicate whether the cable supports one or two lanes of **[USB 3.2]** SuperSpeed signaling.

6.4.4.3.1.7.26 Optically Isolated Active Cable Field

The **Optically Isolated Active Cable** field (B2) **Shall** indicate whether this cable is an optically isolated *Active Cable* or not (as defined in **[USB Type-C 2.4]**). Optically Isolated *Active Cables* **Shall** have a re-timer or linear re-driver (LRD) as the active element and do not support **[USB 2.0]** or carry *VBUS*.

6.4.4.3.1.7.27 USB4 Asymmetric Mode Supported Field

The **USB4 Asymmetric Mode Supported** field (B1) **Shall** indicate that the *Active Cable* supports asymmetric mode as defined in **[USB4]** and **[USB Type-C 2.4]**.

6.4.4.3.1.7.28 USB Gen Field

The **USB Gen** field (B0) **Shall** indicate the signaling Gen the cable supports. Gen 1 **Shall** only be used by **[USB 3.2]** cables as indicated by the USB 3.2 Supported field. Gen 2 or higher **May** be used by either **[USB 3.2]** or **[USB4]** cables as indicated by their respective supported fields. When Gen 2 or higher is indicated the **USB Highest Speed (Active Cable)** field in VDO1 **Shall** indicate the actual Gen supported.

6.4.4.3.1.8 Alternate Mode Adapter VDO

The *Alternate Mode Adapter (AMA) VDO* has been **Deprecated**. *PDUSB Devices* which support one or more *Alternate Modes* **Shall** set an appropriate *Product Type (UFP)*, and **Shall** set the **Modal Operation Supported** bit to '1'.

6.4.4.3.1.9

VCONN Powered USB Device VDO

The *VCONN Powered USB Device* (VPD) VDO defined in this section **Shall** be sent when the *Product Type* is given as *VCONN Powered USB Device*. [Table 6.44, "VPD VDO"](#) defines the VPD VDO which **Shall** be sent.

Table 6.44 VPD VDO

Bit(s)	Field	Description
B31...28	HW Version	0000b...1111b assigned by the VID owner
B27...24	Firmware Version	0000b...1111b assigned by the VID owner
B23...21	VDO Version	Version Number of the VDO (not this specification Version): <ul style="list-style-type: none"> Version 1.0 = 000b Values 001b...111b are Reserved and Shall Not be used.
B20...17	Reserved	Shall be set to zero.
B16...15	Maximum VBUS Voltage	Maximum VPD VBUS Voltage: <ul style="list-style-type: none"> 00b – 20V 01b – 30V¹ (Deprecated) 10b – 40V¹ (Deprecated) 11b – 50V¹ (Deprecated)
B14	Charge Through Current Support	Charge Through Current Support bit=1b: <ul style="list-style-type: none"> 0b - 3A capable. 1b - 5A capable Charge Through Current Support bit = 0b: <ul style="list-style-type: none"> Reserved and Shall be set to zero.
B13	Reserved	Shall be set to zero.
B12...7	VBUS Impedance	Charge Through Current Support bit = 1b: VBUS impedance through the VPD in 2 mΩ increments. Values less than 10 mΩ are Reserved and Shall Not be used. Charge Through Current Support bit = 0b: Reserved and Shall be set to zero.
B6...1	Ground Impedance	Charge Through Current Support bit = 1b: Ground impedance through the VPD in 1 mΩ increments. Values less than 10 mΩ are Reserved and Shall Not be used. Charge Through Current Support bit = 0b: Shall be set to zero.
B0	Charge Through Support	<ul style="list-style-type: none"> 1b – the VPD supports <i>Charge Through</i> 0b – the VPD does not support <i>Charge Through</i>
1) Values no longer allowed. When present the field Shall be interpreted as if it was 00b.		

6.4.4.3.1.9.1

HW Version Field

The **HW Version** field (B31...28) contains a HW version assigned by the VID owner.

6.4.4.3.1.9.2

FW Version Field

The **Firmware Version** field (B27...24) contains a FW version assigned by the VID owner.

6.4.4.3.1.9.3

VDO Version Field

The **VDO Version** field (B23...20) contains a VDO Version for this VDM Version number. This field indicates the expected content for this VDO.

6.4.4.3.1.9.4

Maximum VBUS Voltage Field

The **Maximum VBUS Voltage** field (B16...15) **Shall** contain the maximum voltage that a Sink **Shall Negotiate** through the VPD Charge Through Port as part of an *Explicit Contract*.

Note: The maximum voltage that will be applied to the cable is **vSrcNew** max + **vSrcValid** max. For example, when the **Maximum VBUS Voltage** field is 20V, a *Fixed Supply* of 20V can be *Negotiated* as part of an

Explicit Contract where the absolute maximum voltage that can be applied to the cable is 21.55V.

Maximum VBUS Voltage field values of 01b and 10b (formerly 30V and 40V) **Shall** be treated if they were 00b (20V).

6.4.4.3.1.9.5 V_{BUS} Impedance Field

The **VBUS Impedance** field (B12...7) **Shall** contain the impedance the *VPD* adds in series between the *Source* and the *Sink*. The *Sink* **Shall** take this value into account when requesting current so as to not to exceed the *VBUS IR Drop* limit of 0.5V between the *Source* and itself. If the *Sink* can tolerate a larger *IR Drop* on *VBUS* it **May** do so.

6.4.4.3.1.9.6 Ground Impedance Field

The **Ground Impedance** field (B6...1) **Shall** contain the impedance the *VPD* adds in series between the *Source* and the *Sink*. The *Sink* **Shall** take this value into account when requesting current so as to not to exceed the *Ground IR Drop* limit of 0.25V between the *Source* and itself.

6.4.4.3.1.9.7 Charge Through Field

The **Firmware Version** field (B0) **Shall** be set to 1b when the *VPD* supports *Charge Through* and 0b otherwise.

6.4.4.3.2 Discover SVIDs

The **Discover SVIDs** Command is used by an *Initiator* to determine the SVIDs for which a *Responder* has Modes. The **Discover SVIDs** Command is used in conjunction with the **Discover Modes** Command in the *Discovery Process* to determine which Modes a device supports. The list of SVIDs is always terminated with one or two 0x0000 SVIDs.

The SVID in the **Discover SVIDs** Command **Shall** be set to the PD SID (see "[Table 6.31, "SVID Values"](#)") by both the *Initiator* and the *Responder* for this Command.

The **Number of Data Objects** field in the *Message Header* in the **Discover SVIDs** Command request **Shall** be set to 1 since the **Discover SVIDs** Command request **Shall Not** contain any VDOs.

The **Discover SVIDs** Command **ACK** sent back by the *Responder **Shall** contain one or more SVIDs. The SVIDs are returned 2 per VDO (see [Table 6.45, "Discover SVIDs Responder VDO"](#)). If there are an odd number of supported SVIDs, the **Discover SVIDs** Command is returned ending with a SVID value of 0x0000 in the last part of the last VDO. If there are an even number of supported SVIDs, the **Discover SVIDs** Command is returned ending with an additional VDO containing two SVIDs with values of 0x0000. A *Responder* **Shall** only return SVIDs for which a **Discover Modes** Command request for that SVID will return at least one *Alternate Mode*.*

A *Responder* that does not support any SVIDs **Shall** return a **NAK**.

The **Number of Data Objects** field in the *Message Header* in the **Discover SVIDs** Command **NAK** and **BUSY** responses **Shall** be set to 1 since they **Shall Not** contain any VDOs.

If the *Responder* supports 12 or more SVIDs then the **Discover SVIDs** Command **Shall** be executed multiple times until a **Discover SVIDs** VDO is returned ending either with a SVID value of 0x0000 in the last part of the last VDO or with a VDO containing two SVIDs with values of 0x0000. Each **Discover SVID ACK** Message, other than the one containing the terminating 0x0000 SVID, **Shall** convey 12 SVIDs. The *Responder* **Shall** restart the list of SVIDs each time a **Discover Identity** Command request is received from the *Initiator*.

Note: Since a *Cable Plug* does not retry Messages if the **GoodCRC** Message from the *Initiator* becomes corrupted the *Cable Plug* will consider the **Discover SVIDs** Command **ACK** unsent and will send the same list of SVIDs again.

[Figure 6.18, "Example Discover SVIDs response with 3 SVIDs"](#) shows an example response to the **Discover SVIDs** Command request with two VDOs containing three SVIDs. [Figure 6.19, "Example Discover SVIDs response with 4 SVIDs"](#) shows an example response with two VDOs containing four SVIDs followed by an empty VDO to terminate the response. [Figure 6.20, "Example Discover SVIDs response with 12 SVIDs followed by an empty response"](#) shows an example response with six VDOs containing twelve SVIDs followed by an additional request that returns an empty VDO indicating there are no more SVIDs to return.

Table 6.45 Discover SVIDs Responder VDO

Bit(s)	Field	Description
B31...16	SVID n	16-bit unsigned integer, assigned by the USB-IF or 0x0000 if this is the last VDO and the <i>Responder</i> supports an even number of SVIDs.
B15...0	SVID n+1	16-bit unsigned integer, assigned by the USB-IF or 0x0000 if this is the last VDO and the <i>Responder</i> supports an odd or even number of SVIDs.

Figure 6.18 Example Discover SVIDs response with 3 SVIDs

Header No. of Data Objects = 3	VDM Header	VDO 1		VDO 2	
		SVID 0 (B31..16)	SVID 1 (B15..0)	SVID 2 (B31..16)	0x0000 (B15..0)

Figure 6.19 Example Discover SVIDs response with 4 SVIDs

Header No. of Data Objects = 4	VDM Header	VDO 1		VDO 2		VDO 3	
		SVID 0 (B31..16)	SVID 1 (B15..0)	SVID 2 (B31..16)	SVID 3 (B15..0)	0x0000 (B31..16)	0x0000 (B15..0)

Figure 6.20 Example Discover SVIDs response with 12 SVIDs followed by an empty response

Header No. of Data Objects = 7	VDM Header	VDO 1		VDO 2		VDO 3		VDO 4		VDO 5		VDO 6	
		SVID 0 (B31..16)	SVID 1 (B15..0)	SVID 2 (B31..16)	SVID 3 (B15..0)	SVID 4 (B31..16)	SVID 5 (B15..0)	SVID 6 (B31..16)	SVID 7 (B15..0)	SVID 8 (B31..16)	SVID 9 (B15..0)	SVID 10 (B31..16)	SVID 11 (B15..0)

Header No. of Data Objects = 2	VDM Header	VDO 1	
		0x0000 (B31..16)	0x0000 (B15..0)

6.4.4.3.3 Discover Modes

The **Discover Modes** Command is used by an *Initiator* to determine the Modes a *Responder* supports for a given *SVID*.

The *SVID* in the **Discover Modes** Command **Shall** be set to the *SVID* for which Modes are being requested by both the *Initiator* and the *Responder* for this Command.

The **Number of Data Objects** field in the *Message Header* in the **Discover Modes** Command request **Shall** be set to 1 since the **Discover Modes** Command request **Shall Not** contain any VDOs.

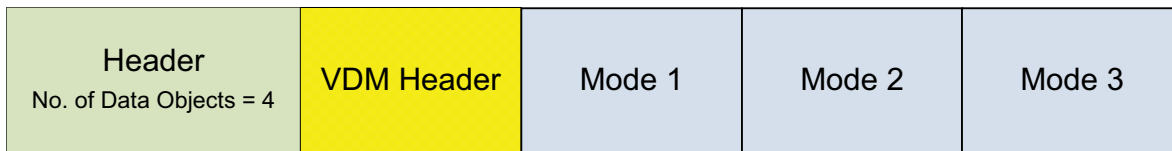
The **Discover Modes** Command **ACK** sent back by the *Responder **Shall** contain one or more Modes. The **Discover Modes** Command **ACK** **Shall** contain a *Message Header* with the **Number of Data Objects** field set to a value of 2 to 7 (the actual value is the number of *Alternate Mode* objects plus one). If the ID is a *VID*, the structure and content of the VDO is left to the Vendor. If the ID is a *SID*, the structure and content of the VDO is defined by the relevant standard's body.*

A *Responder* that does not support any Modes **Shall** return a **NAK**.

The **Number of Data Objects** field in the *Message Header* in the **Discover Modes** Command **NAK** and **BUSY** responses **Shall** be set to 1 since they **Shall Not** contain any VDOs.

[Figure 6.21, "Example Discover Modes response for a given SVID with 3 Modes"](#) shows an example of a **Discover Modes** Command response from a *Responder* which supports three Modes for a given *SVID*.

Figure 6.21 Example Discover Modes response for a given SVID with 3 Modes



6.4.4.3.4 Enter Mode Command

The **Enter Mode** Command is used by an *Initiator* (*DFP*) to command a *Responder* (*UFP* or *Cable Plug*) to enter a specified *Alternate Mode* of operation. Only a *DFP* **Shall** initiate the **Enter Mode** Process which it starts after it has successfully completed the *Discovery Process*.

The value in the Object Position field in the *VDM Header* **Shall** indicate to which *Alternate Mode* in the **Discover Modes** Command the VDO refers (see [Figure 6.21, "Example Discover Modes response for a given SVID with 3 Modes"](#)). The value 1 always indicates the first *Alternate Mode* as it is the first object following the *VDM Header*. The value 2 refers to the next *Alternate Mode* and so forth.

The **Number of Data Objects** field in the *Message Header* in the Command request **Shall** be set to either 1 or 2 since the **Enter Mode** Command request **Shall Not** contain more than 1 VDO. When a VDO is included in an **Enter Mode** Command request the contents of the 32-bit VDO is defined by the *Alternate Mode*.

The **Number of Data Objects** field in the Command response **Shall** be set to 1 since an **Enter Mode** Command response (**ACK**, **NAK**) **Shall Not** contain any VDOs.

Before entering a *Alternate Mode*, by sending the **Enter Mode** Command request that requires the reconfiguring of any pins on entry to that *Alternate Mode*, the *Initiator* **Shall** ensure that those pins being reconfigured are placed into the *USB Safe State*. Before entering an *Alternate Mode* that requires the reconfiguring of any pins, the *Responder* **Shall** ensure that those pins being reconfigured are placed into either *USB operation* or the *USB Safe State*.

A device **May** support multiple Modes with one or more active at any point in time. Any interactions between them are the responsibility of the Standard or Vendor. Where there are multiple *Active Modes* at the same time *Modal Operation* **Shall** start on entry to the first *Alternate Mode*.

On receiving an **Enter Mode** Command requests the *Responder* **shall** respond with either an **ACK** or a **NAK** response. The *Responder* is not allowed to return a **BUSY** response. The value in the Object Position field of the **Enter Mode** Command response **shall** contain the same value as the received **Enter Mode** Command request.

If the *Responder* responds to the **Enter Mode** Command request with an **ACK**, the *Responder* **shall** enter the *Alternate Mode* before sending the **ACK**. The *Initiator* **shall** enter the *Alternate Mode* on reception of the **ACK**. Successful transmission of the *Message* confirms to the *Responder* that the *Initiator* will enter an *Active Mode*.

See [Figure 8.111, "DFP to UFP Enter Mode"](#) for more details.

If the *Responder* responds to the **Enter Mode** Command request with a **NAK**, the *Alternate Mode* is not entered. If not presently in *Modal Operation* the *Initiator* **shall** return to USB operation. If not presently in *Modal Operation* the *Responder* **shall** remain in either USB operation or the *USB Safe State*.

If the *Initiator* fails to receive a response within **tVDMWaitModeEntry** it **shall Not** enter the *Alternate Mode* but return to USB operation.

[Figure 6.22, "Successful Enter Mode sequence"](#) shows the sequence of events during the transition between USB operation and entering an *Alternate Mode*. It illustrates when the *Responder's* *Alternate Mode* changes and when the *Initiator's* *Alternate Mode* changes. [Figure 6.23, "Unsuccessful Enter Mode sequence due to NAK"](#) illustrates that when the *Responder* returns a **NAK** the transition to an *Alternate Mode* do not take place and the *Responder* and *Initiator* remain in their default USB roles.

Figure 6.22 Successful Enter Mode sequence

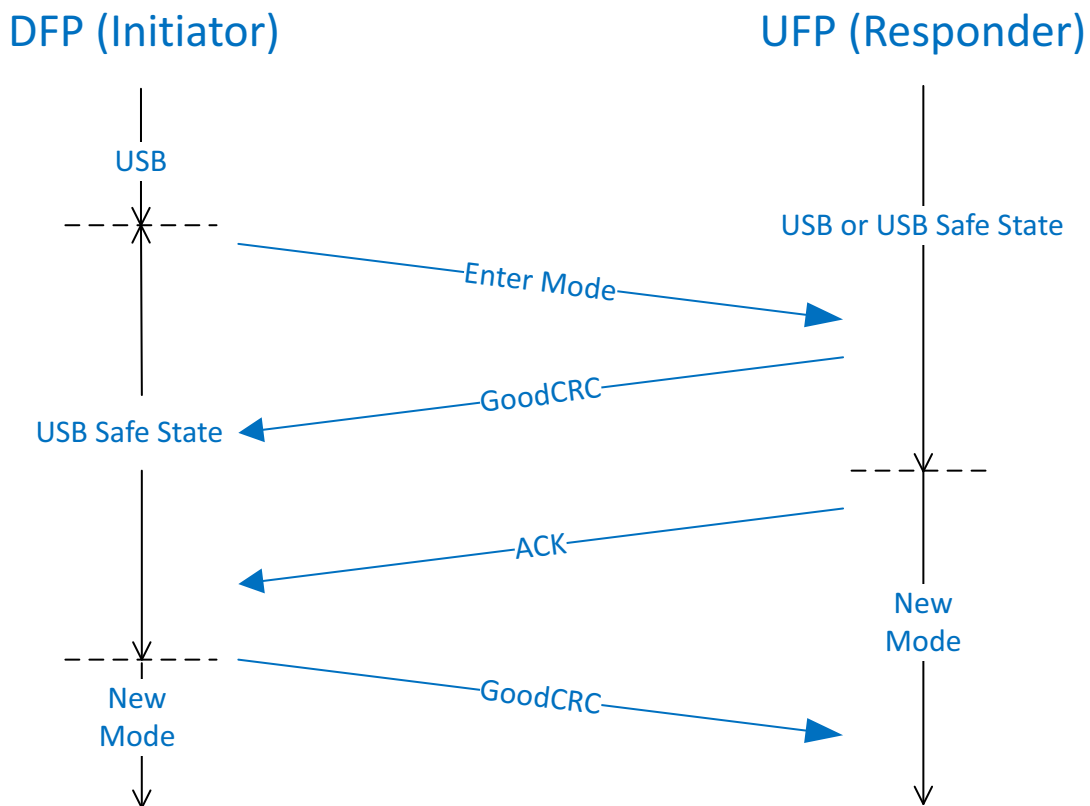
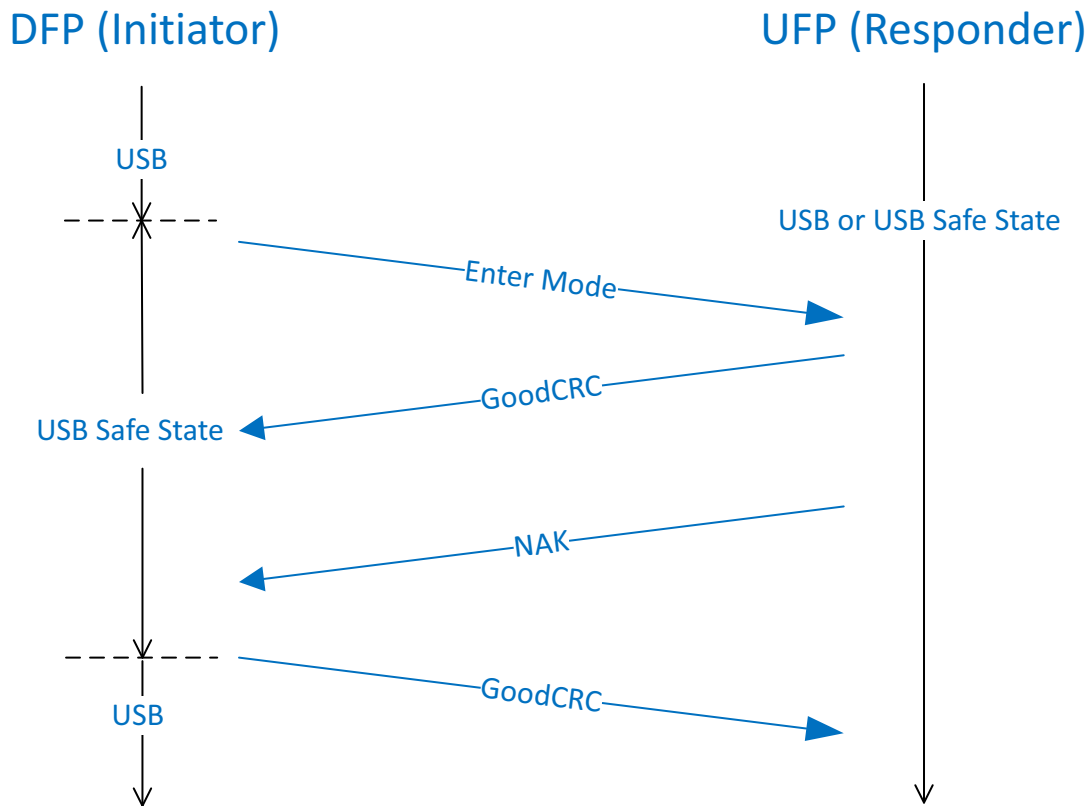


Figure 6.23 Unsuccessful Enter Mode sequence due to NAK



Once the *Alternate Mode* is entered, the device **shall** remain in that *Active Mode* until the *Exit Mode Command* is successful (see [Section 6.4.4.3.5, "Exit Mode Command"](#)).

The following events **shall** also cause the *Port Partners* and *Cable Plug(s)* to exit all *Active Modes*:

- A PD Hard Reset.
- Error Recovery.
- The *Port Partners* or *Cable Plug(s)* are Detached.
- A Cable Reset (only exits the *Cable Plug's Active Modes*).
- A Data Reset (removing power briefly resets all the *Active Modes* in the *Cable Plug*).

The *Initiator shall* return to USB Operation within *tVDMExitMode* of a disconnect, of *Hard Reset Signaling* being detected or *Error Recovery*.

The *Responder shall* return to either USB operation or *USB Safe State* within *tVDMExitMode* of a disconnect, of *Hard Reset Signaling* being detected or *Error Recovery*.

A *DR_Swap Message shall Not* be sent during *Modal Operation* between the *Port Partners* (see [Section 6.3.9, "DR_Swap Message"](#)).

6.4.4.3.5 Exit Mode Command

The *Exit Mode Command* is used by an *Initiator (DFP)* to command a *Responder (UFP or Cable Plug)* to exit its *Active Mode* and return to normal USB operation. Only the *DFP shall* initiate the *Exit Mode Process*.

The value in the Object Position field **shall** indicate to which *Alternate Mode* in the *Discover Modes Command* the VDO refers (see [Figure 6.21, "Example Discover Modes response for a given SVID with 3 Modes"](#)) and **shall** have been used previously in an *Enter Mode Command* request for an *Active Mode*. The value 1 always indicates the first

Alternate Mode as it is the first object following the *VDM Header*. The value 2 refers to the next *Alternate Mode* and so forth. A value of 111b in the Object Position field **Shall** indicate that all *Active Modes* **Shall** be exited.

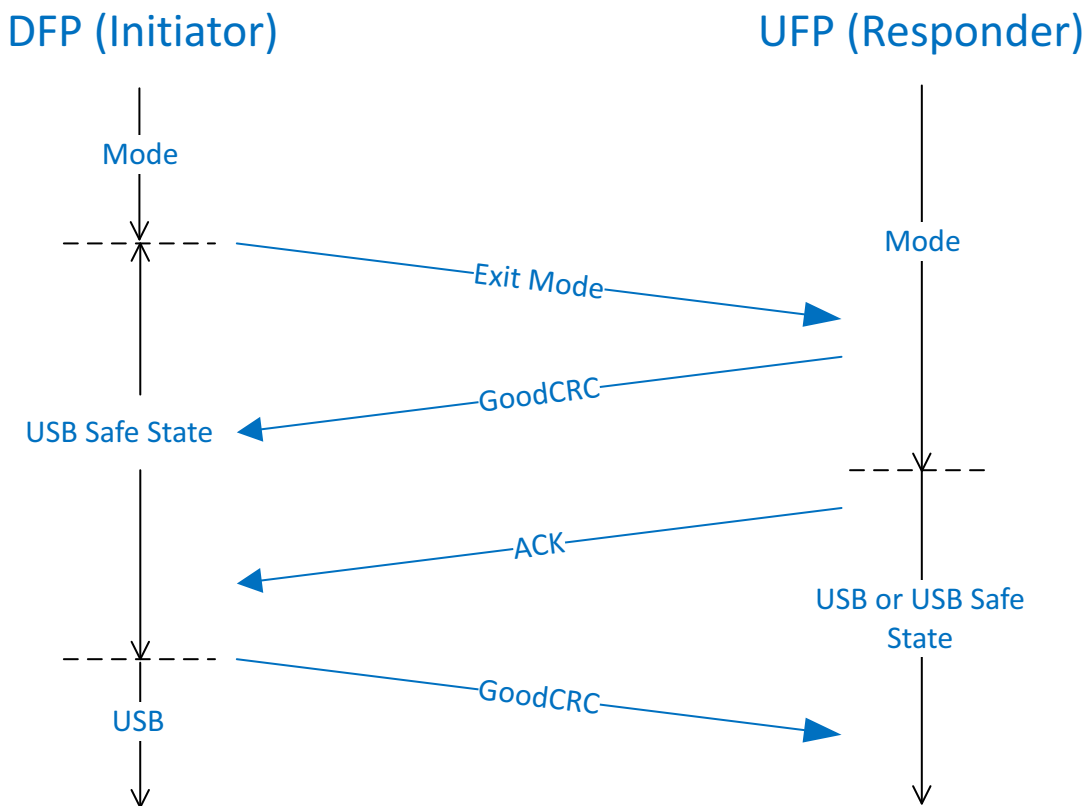
The **Number of Data Objects** field in both the *Command* request and *Command* response (**ACK**, **NAK**) **Shall** be set to 1 since an *Exit Mode Command* **Shall Not** contain any VDOs.

The *Responder* **Shall** exit its *Active Mode* before sending the response *Message*. The *Initiator* **Shall** exit its *Active Mode* when it receives the **ACK**. The *Responder* **Shall Not** return a **BUSY** acknowledgment and **Shall** only return a **NAK** acknowledgment to a request not containing an *Active Mode* (i.e., **Invalid** object position). An *Initiator* which fails to receive an **ACK** within *tVDMWaitModeExit* or receives a **NAK** or **BUSY** response **Shall** exit its *Active Mode*.

See [Figure 8.112, "DFP to UFP Exit Mode"](#) for more details.

[Figure 6.24, "Exit Mode sequence"](#) shows the sequence of events during the transition between exiting an *Active Mode* and USB operation. It illustrates when the *Responder's* *Alternate Mode* changes and when the *Initiator's* *Alternate Mode* changes.

Figure 6.24 Exit Mode sequence



6.4.4.3.6 Attention

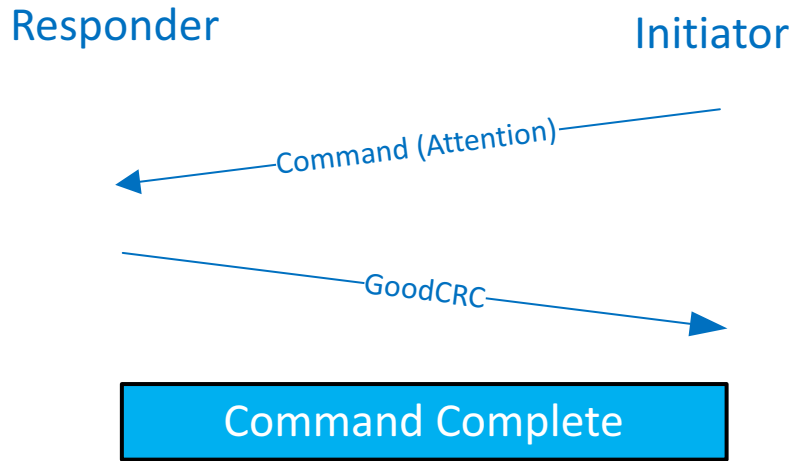
The **Attention** Command **May** be used by the *Initiator* to notify the *Responder* that it requires service.

The value in the Object Position field **Shall** indicate to which *Alternate Mode* in the *Discover Modes* Command the VDO refers (see [Figure 6.21, "Example Discover Modes response for a given SVID with 3 Modes"](#)) and **Shall** have been used previously in an *Enter Mode* Command request for an *Active Mode*. The value 1 always indicates the first *Alternate Mode* as it is the first object following the *VDM Header*. The value 2 refers to the next *Alternate Mode* and so forth. A value of 000b or 111b in the Object Position field **Shall Not** be used by the **Attention** Command.

The **Number of Data Objects** field in the *Message Header* **Shall** be set to 1 or 2 since the **Attention Command** **Shall Not** contain more than 1 VDO. When a VDO is included in an **Attention Command** the contents of the 32-bit VDO is defined by the *Alternate Mode*.

[Figure 6.24, "Exit Mode sequence"](#) shows the sequence of events when an **Attention Command** is received.

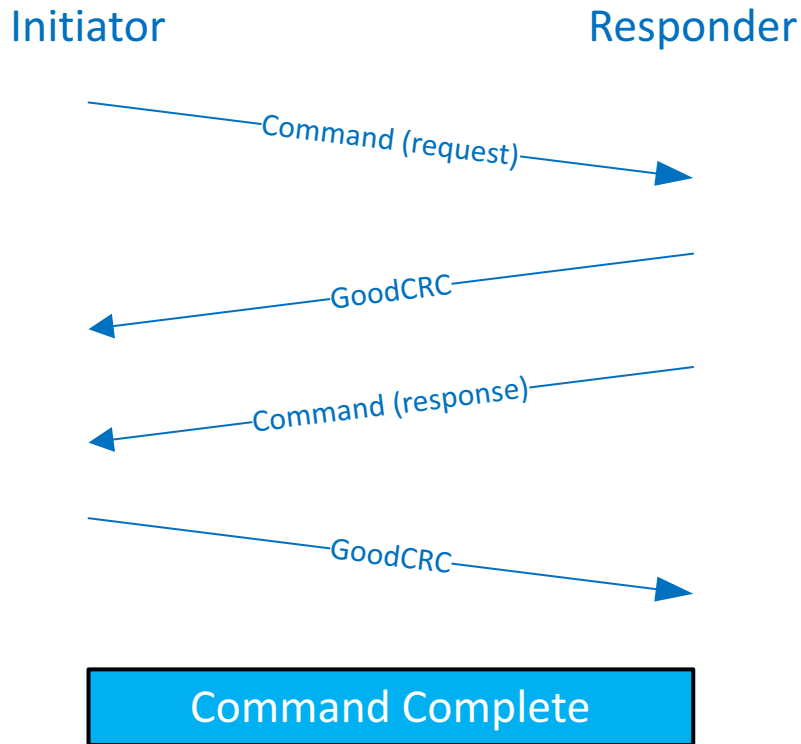
Figure 6.25 Attention Command request/response sequence



6.4.4.4 Command Processes

The *Message flow of Commands* during a Process is a query followed by a response. Every *Command* request sent has to be responded to with a **GoodCRC** Message. The **GoodCRC** Message only indicates the *Command* request was received correctly; it does not mean that the *Responder* understood or even supports a particular *SVID*. [Figure 6.26, "Command request/response sequence"](#) shows the request/response sequence including the **GoodCRC** Messages.

Figure 6.26 Command request/response sequence



In order for the *Initiator* to know that the *Command* request was actually consumed, it needs an acknowledgment from the *Responder*. There are three responses that indicate the *Responder* received and processed the *Command* request:

- *ACK*
- *NAK*
- *BUSY*

The *Responder* **Shall** complete:

- *Enter Mode* requests within *tVDMEnterMode*.
- *Exit Mode* requests within *tVDMExitMode*.
- Other requests within *tVDMReceiverResponse*.

An *Initiator* not receiving a response within the following times **Shall** timeout and return to either the *PE_SRC_Ready* or *PE_SNK_Ready* state (as appropriate):

- *Enter Mode* requests within *tVDMWaitModeEntry*.
- *Exit Mode* requests within *tVDMWaitModeExit*.
- Other requests within *tVDMSenderResponse*.

The *Responder* **Shall** respond with:

- *ACK* if it recognizes the *SVID* and can process it at this time.
- *NAK*:
 - if it recognizes the *SVID* but cannot process the *Command* request

- or if it does not recognize the *SVID*
- or if it does not support the *Command*
- or if a VDO contains a field which is **Invalid**.
- **BUSY** if it recognizes the *SVID* and the *Command* but cannot process the *Command* request at this time.

The **ACK**, **NAK** or **BUSY** response **Shall** contain the same *SVID* as the *Command* request.

6.4.4.4.1 Discovery Process

The *Initiator* (usually the *DFP*) always begins the *Discovery Process*. The *Discovery Process* has two phases. In the first phase, the **Discover SVIDs** *Command* request is sent by the *Initiator* to get the list of *SVIDs* the *Responder* supports. In the second phase, the *Initiator* sends a **Discover Modes** *Command* request for each *SVID* supported by both the *Initiator* and *Responder*.

6.4.4.4.2 Enter Vendor Mode / Exit Vendor Mode Processes

The result of the *Discovery Process* is that both the *Initiator* and *Responder* identify the Modes they mutually support. The *Initiator* (*DFP*), upon finding a suitable *Alternate Mode*, uses the **Enter Mode** *Command* to enable the *Alternate Mode*.

The *Responder* (*UFP* or *Cable Plug*) and *Initiator* continue using the *Active Mode* until the *Active Mode* is exited.

In a managed termination, using the **Exit Mode** *Command*, the *Active Mode* **Shall** be exited in a controlled manner as described in [Section 6.4.4.3.5, "Exit Mode Command"](#).

In an unmanaged termination, triggered by:

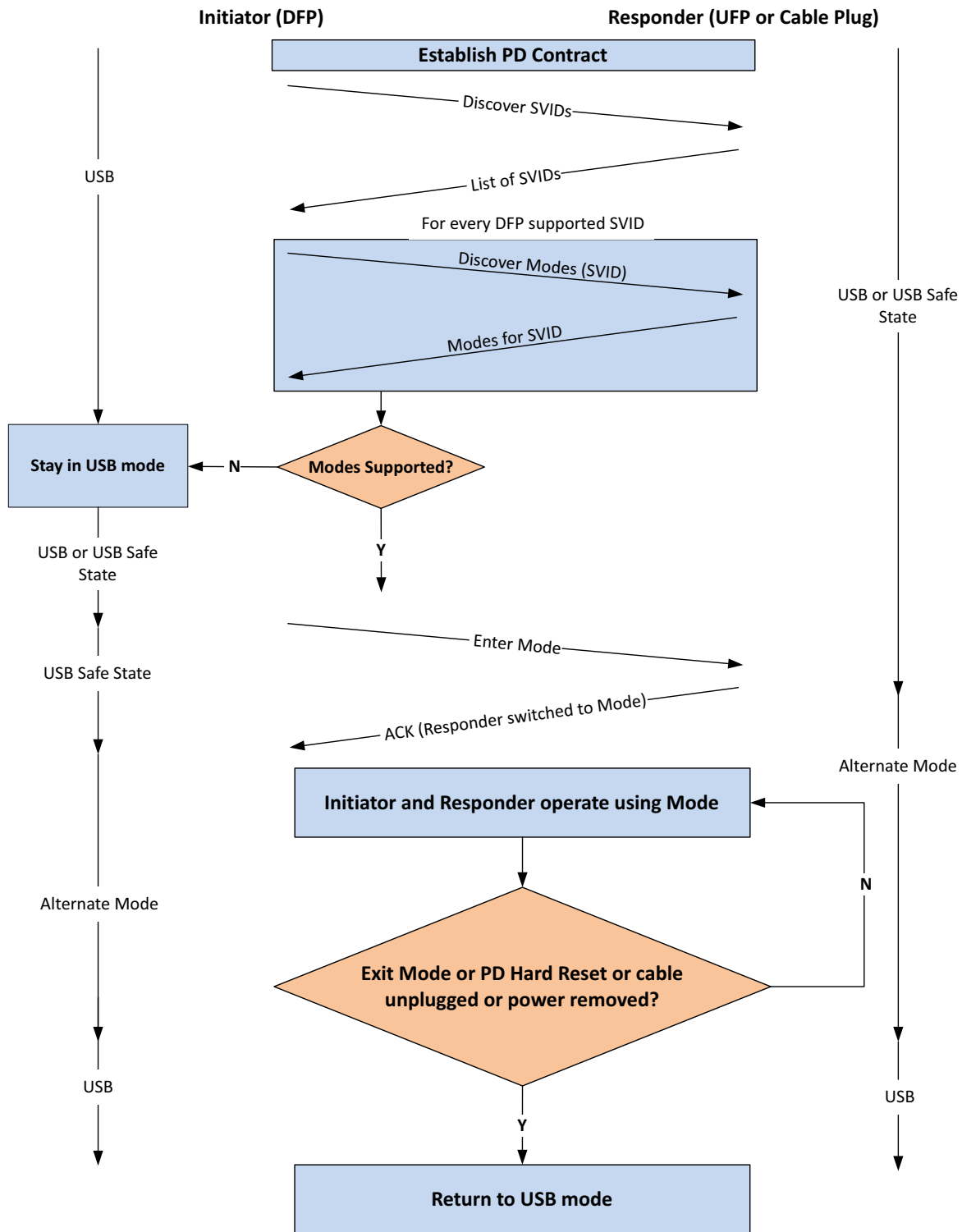
- A Power Delivery *Hard Reset* (i.e. **Hard Reset** *Signaling* sent by either *Port Partner*) or
- By cable *Detach* (device unplugged) or
- By **Error Recovery**

the *Active Mode* **Shall** still be exited but there **Shall Not** be a transition through the *USB Safe State*.

In both the managed and unmanaged terminations, the *Initiator* and *Responder* return to USB operation as defined in [\[USB Type-C 2.4\]](#) following an exit from an *Alternate Mode*.

The overall *Message* flow is illustrated in [Figure 6.27, "Enter/Exit Mode Process"](#).

Figure 6.27 Enter/Exit Mode Process



6.4.4.5 VDM Message Timing and Normal PD Messages

The timing and interspersing of VDMs between regular PD Messages **shall** be done without perturbing the PD AMSs. This requirement **shall** apply to both *Unstructured VDMs* and *Structured VDMs*.

The use of *Structured VDMs* by an *Initiator* **Shall Not** interfere with the normal *PD Message* timing requirements nor **Shall** either the *Initiator* or *Responder* interrupt a *PD AMS* (e.g., *Negotiation*, *Power Role Swap*, *Data Role Swap* etc.). The use of *Unstructured VDMs* **Shall Not** interfere with normal *PD Message* timing.

6.4.5 Battery_Status Message

The **Battery_Status** Message **Shall** be sent in response to a **Get_Battery_Status** Message. The **Battery_Status** Message contains one Battery Status Data Object (BSDO) for one of the Batteries it supports as reported by **Number of Batteries/Battery Slots** field in the **Source_Capabilities_Extended** Message. The returned BSDO **Shall** correspond to the *Battery* requested in the **Battery_Status_Ref** field contained in the **Get_Battery_Status** Message.

The **Battery_Status** Message returns a BSDO whose format **Shall** be as shown in [Figure 6.28, "Battery_Status Message"](#) and [Table 6.46, "Battery_Status Data Object \(BSDO\)"](#). The **Number of Data Objects** field in the **Battery_Status** Message **Shall** be set to 1.

Figure 6.28 Battery_Status Message

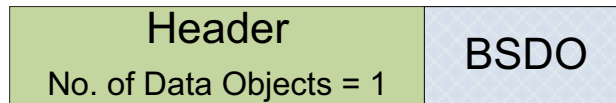


Table 6.46 Battery Status Data Object (BSDO)

Bit(s)	Field	Description	
B31...16	Battery Present Capacity	Battery's State of Charge (SoC) in 0.1 WH increments Note: 0xFFFF = <i>Battery's</i> SOC unknown	
B15...8	Battery Info	Bit	Description
		0	Invalid Battery Reference Invalid <i>Battery</i> reference
		1	Battery Present <i>Battery</i> is present when set
		3...2	Battery Charging Status When Battery Present is '1' Shall contain the <i>Battery</i> charging status: <ul style="list-style-type: none"> 00b: <i>Battery</i> is Charging. 01b: <i>Battery</i> is Discharging. 10b: <i>Battery</i> is Idle. 11b: Reserved, Shall Not be used. When Battery Present is '0': <ul style="list-style-type: none"> 11b...00b: Reserved, Shall Not be used.
B7...0	Reserved	7...4	Reserved, Shall Not be used.
		Shall be set to zero	

6.4.5.1 Battery Present Capacity

The **Battery Present Capacity** field **Shall** return either the *Battery's* State of Charge (SoC) in tenths of WH or indicate that the *Battery's* present State of Charge (SOC) is unknown.

6.4.5.2 Battery Info

The **Battery Info** field **shall** be used to report additional information about the *Battery*'s present status. The **Battery Info** field's bits **shall** reflect the present conditions under which the *Battery* is operating in the systems.

6.4.5.2.1 Invalid Battery Reference

The **Invalid Battery Reference** bit **shall** be set when the *Get_Battery_Status* Message contains a reference to a *Battery* or *Battery Slot* (see [Section 6.5.1.13, "Number of Batteries/Battery Slots Field"](#)) that does not exist.

6.4.5.2.2 Battery Present

The **Battery Present** bit **shall** be set whenever the *Battery* is present. It **shall** always be set for Batteries that are not Hot Swappable Batteries. For Hot Swappable Batteries, the **Battery Present** bit **shall** indicate whether the *Battery* is *Attached* or *Detached*.

6.4.5.2.3 Battery Charging Status

The **Battery Charging Status** bits indicate whether the *Battery* is being charged, discharged or is idle (neither charging nor discharging). These bits **shall** be set when the **Battery Present** bit is set. Otherwise, when the **Battery Present** bit is zero the **Battery Charging Status** bits **shall** also be zero.

6.4.6 Alert Message

The **Alert Message** is provided to allow *Port Partners* to inform each other when there is a status change event. Some of the events are critical such as *OCP*, *OVP* and *OTP*, while others are informational such as change in a *Battery*'s status from charging to neither charging nor discharging.

The **Alert Message** **Shall** only be sent when the *Source* or *Sink* detects a status change.

The **Alert Message** **Shall** contain exactly one Alert Data Object (ADO) and the format **Shall** be as shown in [Figure 6.29, "Alert Message"](#) and [Table 6.47, "Alert Data Object \(ADO\)"](#).

Figure 6.29 Alert Message

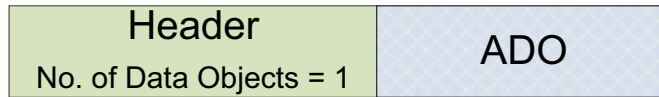


Table 6.47 Alert Data Object (ADO)

Bit(s)	Field	Description	
		Bit	Description
B31...24	Type of Alert	0	Reserved and Shall be set to zero.
		1	Battery Status Change Event Battery Status Change Event (<i>Attach/Detach</i> /charging/discharging/idle)
		2	OCP Event OCP event when set (<i>Source</i> only, for <i>Sink</i> Reserved and Shall be set to zero).
		3	OTP Event OTP event when set
		4	Operating Condition Change Operating Condition Change when set
		5	Source Input Change Event Source Input Change Event when set
		6	OVP Event OVP event when set
		7	Extended Alert Event Extended Alert Event when set
B23...20	Fixed Batteries	When Battery Status Change Event bit set indicates which <i>Fixed Batteries</i> have had a status change. B20 corresponds to <i>Battery 0</i> and B23 corresponds to <i>Battery 3</i> .	
B19...16	Hot Swappable Batteries	When Battery Status Change Event bit set indicates which <i>Hot Swappable Batteries</i> have had a status change. B16 corresponds to <i>Battery 4</i> and B19 corresponds to <i>Battery 7</i> .	

Table 6.47 Alert Data Object (ADO) (Continued)

Bit(s)	Field	Description
B15...4	Reserved	Shall be set to zero
B3...0	Extended Alert Event Type	<p>When the Extended Alert Event bit in the Type of Alert field equals '1', then the Extended Alert Event Type field indicates the event which has occurred:</p> <ul style="list-style-type: none"> 0 = Reserved. 1 = Power state change (<i>DFP</i> only) 2 = Power button press (<i>UFP</i> only) 3 = Power button release (<i>UFP</i> only) 4 = Controller initiated wake e.g., Wake on LAN (<i>UFP</i> only) 5-15 = Reserved <p>When the Extended Alert Event bit in the Type of Alert field equals '0', then the Extended Alert Event Type field is Reserved and Shall be set to zero.</p>

6.4.6.1 Type of Alert

The **Type of Alert** field **Shall** be used to report *Source* or *Sink* status changes. Only one **Alert Message** **Shall** be generated for each Event or Change; however multiple Type of Alert bits **May** be set in one **Alert Message**. Once the **Alert Message** has been sent the **Type of Alert** field **Shall** be cleared.

A **Get_Battery_Status** Message **Should** be sent in response to a *Battery* status change in an **Alert Message** to get the details of the change.

A **Get_Status** Message **Should** be sent in response to a non-*Battery* status change in an **Alert Message** from to get the details of the change.

6.4.6.1.1 Battery Status Change

The **Battery Status Change Event** bit **Shall** be set when any *Battery*'s power state changes between charging, discharging, neither. For Hot Swappable Batteries, it **Shall** also be set when a *Battery* is *Attached* or *Detached*.

6.4.6.1.2 Over-Current Protection Event

The **OCP Event** bit **Shall** be set when a *Source* detects its output current exceeds its limits triggering its protection circuitry. This bit is **Reserved** for a *Sink*.

6.4.6.1.3 Over-Temperature Protection Event

The **OTP Event** bit **Shall** be set when a *Source* or *Sink* shuts down due to over-temperature triggering its protection circuitry.

6.4.6.1.4 Operating Condition Change

The **Operating Condition Change** bit **Shall** be set when a *Source* or *Sink* detects its Operating Condition enters or exits either the 'warning' or 'over temperature' temperature states.

The **Operating Condition Change** bit **Shall** be set when the *Source* operating in the Programmable Power Supply mode detects it has changed its operating condition between *Constant Voltage (CV)* and *Current Limit (CL)*.

6.4.6.1.5 Source Input Change Event

The **Source Input Change Event** bit **Shall** be set when the *Source/Sink*'s input changes. For example, when the AC input is removed, and the *Source/Sink* continues to be powered from one or more of its batteries or when AC returns and the *Source/Sink* transitions from *Battery* to AC operation or when the *Source/Sink* changes operation from one (or more) *Battery* to another (or more) *Battery*.

6.4.6.1.6 Over-Voltage Protection Event

The **OVP Event** bit **Shall** be set when the *Sink* detects its output voltage exceeds its limits triggering its protection circuitry.

The **OVP Event** bit **May** be set when the *Source* detects its output voltage exceeds its limits triggering its protection circuitry.

6.4.6.1.7 Extended Alert Event

The **Extended Alert Event** bit **Shall** be set when the event is defined as an Extended Alert Type.

6.4.6.2 Fixed Batteries

The **Fixed Batteries** field indicates which *Fixed Batteries* have had a status change. B20 corresponds to *Battery 0* and B23 corresponds to *Battery 3*.

Once the **Alert Message** has been sent the **Fixed Batteries** field **Shall** be cleared.

6.4.6.3 Hot Swappable Batteries

The **Hot Swappable Batteries** field indicates which Hot Swappable Batteries have had a status change. B16 corresponds to *Battery 0* and B19 corresponds to *Battery 3*.

Once the **Alert Message** has been sent the **Hot Swappable Batteries** field **Shall** be cleared.

6.4.6.4 Extended Alert Event Types

The **Extended Alert Event Type** field provides extensions to the available types for the **Alert Message**. If the **Extended Alert Event Type** bit is not set, then the Extended Alert Event Type is **Reserved** and **Shall** be set to zero.

6.4.6.4.1 Power State Change

The Power state change event value **May** be set when the *DFP* transitions into a new power state. The new power state **Shall** be communicated via the Power state change byte in the **Status Message**. This **Message Should** be sent by the host in response to any system power state change.

6.4.6.4.2 Power Button Press

The Power button press event value **May** be set when the power button on the *UFP* is pressed. The press and release events are separated into two different events so that devices that respond differently to a long button press will see a long button press. On the host-side, the power button press event typically initiates the same behavior as a power button press of the host's power button.

6.4.6.4.3 Power Button Release

If a Power button press event was sent, then the Power button release event value **Shall** be sent by the *UFP* following the Power button press event. If a physical power button press initiated the Power button press event, then the Power button release event **Should** be sent when the physical button is released.

6.4.6.4.4 Controller Initiated Wake

The Controller initiated wake is used to communicate a wake event from the *UFP* to the DPF such as Wake on LAN from a NIC or another controller. This event doesn't need the press/release form of the Power button press, because it only needs to communicate the presence of the event, and not the timing.

6.4.7 Get_Country_Info Message

The **Get_Country_Info** Message **Shall** be sent by a *Port* to get country specific information from its *Port Partner* using the country's Alpha-2 Country Code defined by [\[ISO 3166\]](#). The *Port Partner* responds with a **Country_Info** Message that contains the country specific information. The **Get_Country_Info** Message **Shall** be as shown in [Figure 6.30, "Get_Country_Info Message"](#) and [Table 6.48, "Country Code Data Object \(CCDO\)"](#).

For example, if the request is for China information, then the Country Code Data Object (CCDO) would be CCDO [31:0] = 434E0000h for "CN" country code.

Figure 6.30 Get_Country_Info Message

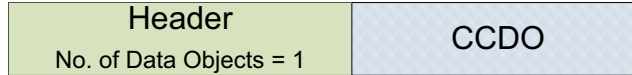


Table 6.48 Country Code Data Object (CCDO)

Bit(s)	Description
B31...24	First character of the Alpha-2 Country Code defined by [ISO 3166]
B23...16	Second character of the Alpha-2 Country Code defined by [ISO 3166]
B15...0	Reserved, Shall be set to zero.

6.4.8 Enter_USB Message

The **Enter_USB** Message **Shall** be sent by the DFP to its UFP Port Partner and to the Cable Plug(s) of an Active Cable, when in an *Explicit Contract*, to enter a specified USB Mode of operation. The recipient of the Message **Shall** respond by sending an **Accept** Message, a **Wait** Message or a **Reject** Message (see [Section 6.9, "Accept, Reject and Wait"](#)).

When entering **[USB4]** operation, the **Enter_USB** Message **Shall** be sent by a **[USB4]** PDUSB Hub's DFP(s) or **[USB4]** PDUSB Host's DFP(s) within **tEnterUSB**:

- following a PD Connection.
- after a Data Reset to enter **[USB4]** operation is completed.
- after a Data Role Swap is completed.

The **Enter_USB** Message **May** be sent by a PDUSB Hub's DFP(s) or PDUSB Host's DFP(s) within **tEnterUSB** following a PD Connection or after a Data Reset to enter **[USB 3.2]** or **[USB 2.0]** operation.

The **Enter_USB** Message **Shall** be used by a PDUSB Hub's DFP(s) to speculatively train the USB links or enter **[DPTC2.1]** or **[TBT3]** Alternate Modes prior to the presence of a host. In this case, the **Host Present** bit **Shall** be cleared. When the Host is Connected the **Enter_USB** Message **Shall** be resent with the **Host Present** bit set. The **Enter_USB** Message's Enter USB Data Object (EUDO), received from the Root Hub when the USB Host is connected, **Shall** be propagated down through the Hub tree.

See **[USB Type-C 2.4]** USB4[®] Hub Connection Requirements.

The **Enter_USB** Message **Shall** be as shown in [Figure 6.31, "Enter_USB Message"](#) and [Table 6.49, "Enter_USB Data Object \(EUDO\)"](#).

Figure 6.31 Enter_USB Message

Header	EUDO
No. of Data Objects = 1	

Table 6.49 Enter_USB Data Object (EUDO)

Bit(s)	Field	Description
B31	Reserved	Shall be set to zero.
B30...28	USB Mode ¹	<ul style="list-style-type: none"> 000b: 001b: 010b: 111b...011b: Reserved, Shall Not be used.
B27	Reserved	Shall be set to zero.
B26	USB4 DRD ²	<ul style="list-style-type: none"> 0b: Not capable of operating as a [USB4] Device 1b: Capable of operating as a [USB4] Device
B25	USB3 DRD ²	<ul style="list-style-type: none"> 0b: Not capable of operating as a [USB 3.2] Device 1b: Capable of operating as a [USB 3.2] Device
B24	Reserved	Shall be set to zero.
B23...21	Cable Speed ^{2,3}	<ul style="list-style-type: none"> 000b: [USB 2.0] only, no SuperSpeed support 001b: [USB 3.2] Gen1 010b: [USB 3.2] Gen2 and [USB4] Gen2 011b: [USB4] Gen3 100b: [USB4] Gen4 101b...111b: Reserved, Shall Not be used.
B20...19	Cable Type ^{2,3}	<ul style="list-style-type: none"> 00b: Passive 01b: Active Re-timer 10b: Active Re-driver 11b: Optically Isolated
B18...17	Cable Current ²	<ul style="list-style-type: none"> 00b = V_{BUS} is not supported 01b = Reserved 10b = 3A 11b = 5A
B16	PCIe Support ²	[USB4] PCIe tunneling supported by the host
B15	DP Support ²	[USB4] DP tunneling supported by the host
B14	TBT Support ²	[TBT3] is supported by the host's USB4 [®] Connection Manager
B13	Host Present ²	<p>A <i>Host</i> is present at the top of the USB tree.</p> <p>When this bit is set PCIe Support, DP Support and TBT Support represent the <i>Host's Capabilities</i> that Shall be propagated down the <i>Hub</i> tree.</p>
B12...0	Reserved	Shall be set to zero.
<p>1) Entry into [USB 3.2] and [USB4] include entry into [USB 2.0].</p> <p>2) Shall be Ignored when received by a <i>Cable Plug</i> (e.g., <i>SOP'</i> or <i>SOP''</i>).</p> <p>3) The <i>DFP</i> Shall interpret the <i>Cable Plug's</i> reported capability as defined in [USB Type-C 2.4] in the USB4 Discovery and Entry Section.</p>		

6.4.8.1 USB Mode Field

The **USB Mode** field **Shall** be used by the *DFP* to direct the *USB Mode* the *Port Partner* is to enter.

6.4.8.2 USB4® DRD Field

The **USB4 DRD** field **Shall** be set when the *Host DFP* is capable of operating as a **[USB4]** Device. A **[USB4]** *Host DFP* that sets the **USB4 DRD** field **Shall** also be capable of operating as a **[USB 2.0]** Device.

6.4.8.3 USB3 DRD Field

The **USB3 DRD** field **Shall** be set when the *Host DFP* is capable of operating as a **[USB 3.2]** Device. A **[USB 3.2]** *Host DFP* that sets the **USB3 DRD** field **Shall** also be capable of operating as a **[USB 2.0]** Device.

6.4.8.4 Cable Speed Field

The **Cable Speed** field **Shall** be used to indicate the cable's maximum speed. The value is read from the *Cable Plug* and interpreted by the *DFP* as defined by **[USB Type-C 2.4]** in the USB4 Discovery and Entry Section.

6.4.8.5 Cable Type Field

The **Cable Type** field **Shall** be used to indicate whether the cable is passive or active. Further if the cable is active, it indicates the type of active circuits in the cable and if the cable is optically isolated. The value is read from the *Cable Plug* and interpreted by the *DFP* as defined by **[USB Type-C 2.4]** in the USB4 Discovery and Entry Section.

6.4.8.6 Cable Current Field

The **Cable Current** field **Shall** be used to indicate the cable's current carrying capability. The value is reported by the *Cable Plug* in the **Vbus Current Handling Capability (Passive Cable)/Vbus Current Handling Capability (Active Cable)** field.

6.4.8.7 PCIe Support Field

The **PCIe Support** field **Shall** be set when the *Host DFP* is capable of tunneling PCIe over **[USB4]**.

The **PCIe Support** field **May** be set speculatively when the *Hub's DFP* is capable of tunneling PCIe over **[USB4]**.

6.4.8.8 DP Support Field

The **DP Support** field **Shall** be set when the *Host DFP* is capable of tunneling DP over **[USB4]**.

The **DP Support** field **May** be set speculatively when the *Hub's DFP* is capable of tunneling DP over **[USB4]**.

6.4.8.9 TBT Support Field

The **TBT Support** field **Shall** be set when the *Host DFP* is capable of tunneling Thunderbolt™ over **[USB4]** and that the Connection Manager (CM) supports discovery and configuration of Thunderbolt 3 devices connected to the *DFP* of **[USB4]** Hubs.

The **TBT Support** field **May** be set speculatively when the *Hub's DFP* is capable of tunneling Thunderbolt over **[USB4]**.

6.4.8.10 Host Present Field

The **Host Present** field **Shall** be set to indicate that a *Host* is present upstream.

6.4.9 EPR_Request Message

An **EPR_Request** Message **Shall** be sent by a Sink, operating in *EPR Mode*, to request power, typically during the request phase of a power *Negotiation*. The **EPR_Request** Message **Shall** be sent in response to the most recent **EPR_Source_Capabilities** Message. The **EPR_Request** Message **Shall** return a Sink Request Data Object (RDO) that **Shall** identify the Power Data Object being requested followed by a copy of the Power Data Object being requested.

Note: The requested Power Data Object **May** be either an EPR (A)PDO or SPR (A)PDO.

The **EPR_Request** Message **Shall** be as shown in [Figure 6.32, "EPR_Request Message"](#).

Figure 6.32 EPR_Request Message

Header	RDO	Copy of PDO
No. of Data Objects = 2		

The Source **Shall** verify the PDO in the **EPR_Request** Message exactly matches the PDO in the latest **EPR_Source_Capabilities** Message pointed to by the Object Position field in the RDO.

The Source **Shall** respond to an **EPR_Request** Message in the same manner as it responds to a **Request** Message with an **Accept** Message, or a **Reject** Message (see [Section 6.9, "Accept, Reject and Wait"](#)). The *Explicit Contract Negotiation* process for EPR is the same as the process for *SPR Mode* except that the **Source_Capabilities** Message is replaced by the **EPR_Source_Capabilities** and the **Request** Message is replaced by the **EPR_Request** Message.

An EPR Source operating in *SPR Mode* that receives a **EPR_Request** Message **Shall** initiate a *Hard Reset*.

The RDO takes a different form depending on the kind of power requested. The PDO and APDO formats are detailed in [Section 6.4.2, "Request Message"](#).

6.4.10 EPR_Mode Message

The **EPR_Mode** Message is used to enter, acknowledge, and exit the *EPR Mode*. The Action field is used to describe the action that is to be taken by the recipient of the **EPR_Mode** Message. The Data field provides additional information for the Message recipient in the *EPR Mode* Data Object (ERMDO).

The **EPR_Mode** Message **shall** be as shown in [Figure 6.33, "EPR Mode DO Message"](#) and [Table 6.50, "EPR Mode Data Object \(ERMDO\)"](#).

Figure 6.33 EPR Mode DO Message

Header	EPRMDO
No. of Data Objects = 1	

Table 6.50 EPR Mode Data Object (ERMDO)

Bit(s)	Field	Description		
B31...24	Action	Value	Action	Sent By
		0x00	Reserved and Shall Not be used.	
		0x01	Enter	Sink only
		0x02	Enter Acknowledged	Source only
		0x03	Enter Succeeded	Source only
		0x04	Enter Failed	Source only
		0x05	Exit	Sink or Source
		0x06...0xFF	Reserved and Shall Not be used.	
B23...16	Data	Action Field	Data Field Value	
		Enter	Shall be set to the <i>EPR Sink</i> Operational PDP	
		Enter Acknowledged	Shall be set to zero	
		Enter Succeeded	Shall be set to zero	
		Enter Failed	Shall be one of the following values: <ul style="list-style-type: none">0x00 - Unknown cause0x01 - Cable not <i>EPR Capable</i>0x02 –Source failed to become <i>VCONN Source</i>.0x03 – <i>EPR Capable</i> bit not set in <i>RDO</i>.0x04 – Source unable to enter <i>EPR Mode</i>¹.0x05 - <i>EPR Capable</i> bit not set in <i>PDO</i>. All other values are Reserved and Shall Not be used	
		Exit	Shall be set to zero	
B15...0	Reserved	Shall be set to zero		
1) The Sink May retry entering <i>EPR Mode</i> after receiving this Enter Failed response.				

6.4.10.1 Process to enter EPR Mode

An *EPR Source* **shall** enter *EPR Mode* upon request by an *EPR Sink* connected with an *EPR Cable* when able to offer the *Source Capabilities* as defined in the *Power Rules* (See [Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable"](#) and [Table 10.13, "EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable"](#)).

For *Port Partners* to successfully enter *EPR Mode*, the following conditions must be met:

- The Sink **shall** request entry into the *EPR Mode*.

- The Source **Shall** verify the cable is *EPR Capable*.
- A Sink **Shall Not** be Connected to the Source through a *Charge Through VPD (CT-VPD)*.
- The Source and Sink **Shall** already be in an *SPR Explicit Contract*.
- The *EPR Capable* bit **Shall** be set in the *Fixed Supply 5V PDO*.
- The *EPR Capable* bit **Shall** have been set in the *RDO* in the last *Request* Message received by the Source.


To verify the cable is *EPR Capable*, the *EPR Source* **Shall** have already done the following (see [Section 6.6.21.4, "tEPRSourceCableDiscovery"](#)):

- Discover the cable prior to entering its *First Explicit Contract*
- Alternatively, within *tEPRSourceCableDiscovery* of entry into the *First Explicit Contract*
 - If it is the *VCONN Source*, discover the cable.
 - If not the *VCONN Source*, do a *VCONN Swap* then discover the cable.

and can verify the cable is *EPR Capable* by completing steps [5](#) and [6](#) in the entry process in [Figure 6.34, "Illustration of process to enter EPR Mode"](#).

The *EPR Mode* entry process is a *Non-interruptible multi-Message AMS*. An illustration of this *AMS* is shown in [Figure 6.34, "Illustration of process to enter EPR Mode"](#).

Note: [Figure 6.34, "Illustration of process to enter EPR Mode"](#) is not **Normative** but is **Informative** only.



SPR Mode



- 1) The Sink **shall** send the **EPR_Mode** Message with the Action field set to 1 (**Enter**) and the Data field set to its Operational PDP. If the EPR Source receives an **EPR_Mode** Message with the Action field not set to **Enter** it **shall** initiate a Soft Reset.

- 2) The *Source* **shall** do the following:

- a) Verify the **EPR Capable** bit was set in the most recent *RDO*. If not set, the *Source* **Shall** do the following:
 - i) Send an **EPR_Mode** Message with the Action field set to 4 (**Enter Failed**) and the Data field set to 3 ("EPR Mode Capable bit not set in the RDO").
 - ii) Abort the *EPR Mode* entry process and remain in the existing SPR *Explicit Contract*.
 - b) Verify the **EPR Capable** bit was set in the most recent 5V Fixed Supply *PDO*. If not set, the *Source* **Shall** do the following:
 - i) Send an **EPR_Mode** Message with the Action field set to 4 (**Enter Failed**) and the Data field set to 5 ("EPR Mode Capable bit not set in the Fixed Supply 5V *PDO*").
 - ii) Abort the *EPR Mode* entry process and remain in the existing SPR *Explicit Contract*.
 - c) Verify the *Source* is still able to support *EPR Mode*. If not, the *Source* **Shall** do the following:
 - i) Send an **EPR_Mode** Message with the Action field set to 4 (**Enter Failed**) and Data field set to 4 ("Unable at this time").
 - ii) Abort the *EPR Mode* entry process and remain in the existing SPR *Explicit Contract*. The *Sink* **May** at some time in the future send another request to enter *EPR Mode*.
 - d) Send an **EPR_Mode** Message with the Action field set to 2 (**Enter Acknowledged**).
- 3) If the *Sink* receives any Message, other than an **EPR_Mode** Message with the Action Field set to 2, the *Sink* **Shall** initiate a *Soft Reset*.
 - 4) When the *EPR Source* has used the **Discover Identity** Command to determine and remembers the *Cable Capabilities* or the *EPR Source* is connected with a captive cable:
 - a) If the cable is *EPR Capable* it **Should** go directly to Step 7, but **May** continue to Step 5.
 - b) If the cable is not *EPR Capable* it **Shall** do the following:
 - c) Send an **EPR_Mode** Message with the Action field set to 4 (**Enter Failed**) and the Data field set to 1 ("Cable not EPR capable").
 - d) Abort the *EPR Mode* entry process and remain in the existing SPR *Explicit Contract*.
 - 5) If the *Source* is not the *VCONN Source*, it **Shall** send a **VCONN_Swap** Message
 - a) If the *Source* fails to become the *VCONN Source*, it **Shall**:
 - i) Send an **EPR_Mode** Message with the Action field set to 4 (**Enter Failed**) and the Data field set to 2 (not *VCONN Source*).
 - ii) Abort the *EPR Mode* entry process and remain in the existing SPR *Explicit Contract*.
 - 6) The *Source* **Shall** use the **Discover Identity** Command to read the cable's e-Marker and verify the following:
 - a) Cable VDO - **Maximum VBUS Voltage (Passive Cable)/Maximum VBUS Voltage (Active Cable)** field is 11b (50V)
 - b) Cable VDO - **VBUS Current Handling Capability (Passive Cable)/VBUS Current Handling Capability (Active Cable)** field is 10b (5A)
 - c) Cable VDO - **EPR Capable (Passive Cable)/EPR Capable (Active Cable)** field is 1b (*EPR Capable*)
 - d) If the cable fails to respond to the **Discover Identity** Command or is not *EPR Capable*, the *Source* **Shall** do the following:
 - i) Send an **EPR_Mode** Message with the Action field set to 4 (**Enter Failed**) and the Data field to 1 ("Cable not EPR capable").

- ii) Abort the *EPR Mode* entry process and remain in the existing *SPR Explicit Contract*.
- 7) The *Source* **Shall** send the *EPR_Mode* Message with the Action field set to 3 (*Enter Succeeded*) and **Shall** enter *EPR Mode*.
- 8) If the *Sink* receives an *EPR_Mode* Message with the Action field set to 3 (*Enter Succeeded*) it **Shall** enter *EPR Mode*, otherwise it **Shall** initiate a *Soft Reset*.

If the *EPR Mode* entry process successfully completes within *tEnterEPR* of the last bit of the *GoodCRC* Message sent in response to the *EPR_Mode* Message with the Action field set to 1 (*Enter*), the *Source* **Shall** send an *EPR_Source_Capabilities* Message within *tFirstSourceCap*.

If the *EPR Mode* entry process has not been aborted or does not complete within *tEnterEPR* of the last bit of the *GoodCRC* Message sent in response to the *EPR_Mode* Message with the Action field set to 1 (*Enter*), the *Sink* **Shall** initiate a *Soft Reset*.

6.4.10.2 Operation in EPR Mode

While operating in *EPR Mode*, the *Source* **Shall** only send *EPR_Source_Capabilities* Messages to Advertise its power Capabilities and the *Sink* **Shall** only respond with *EPR_Request* Messages to Negotiate Explicit Contracts. The *EPR_Request* Message **May** be for either an *SPR* or *EPR (A)PDO*.

If the *Source* sends a *Source_Capabilities* Message, that is not in response to a *Sink Get_Source_Cap* Message, the *Sink* **Shall** initiate a *Hard Reset*. If the *Sink* sends a *Request* Message, the *Source* **Shall** initiate a *Hard Reset*.

The *Source* **Shall** monitor the *CC* communications path to ensure that there is periodic traffic. The *Sink* **Shall** send an *EPR_KeepAlive* Message when it has not sent any Messages for more than *tSinkEPRKeepAlive* to ensure there is timely periodic traffic. If there is no traffic for more than *tSourceEPRKeepAlive*, the *Source* **Shall** initiate a *Hard Reset*.

6.4.10.3 Exiting EPR Mode

6.4.10.3.1 Commanded Exit

While in *EPR Mode*, either the *Source* or *Sink* **May** exit *EPR Mode* by sending an *EPR_Mode* Message with the Action field set to 5 (*Exit*).

The ports **Shall** be in an *Explicit Contract* with an *SPR (A)PDO* prior to the *EPR Mode* exit process by either:

- The *Source* sending an *EPR_Source_Capabilities* Message with no *EPR (A)PDO* s (e.g., only *SPR (A)PDO* s) or
- The *Sink* negotiating a new *Explicit Contract* with bit 31 in the *RDO* set to zero (e.g., only *SPR (A)PDO* s)).

The process to exit *EPR Mode* is a *Non-interruptible* multi-Message *AMS* and **Shall** follow these steps in order:

- 1) The *Port Partners* **Shall** be in an *Explicit Contract* with an *SPR (A)PDO*.
- 2) Either the *Source* or *Sink* **Shall** send an *EPR_Mode* Message with the Action field set to 5 (*Exit*) to exit the *EPR Mode*
- 3) The *Source* **Shall** send a *Source_Capabilities* Message within *tFirstSourceCap* of the *GoodCRC* Message in response to the *EPR_Mode* Message with the Action field set to 5 (*Exit*).
- 4) If the *Sink* does not receive a *Source_Capabilities* Message within *tTypeCSinkWaitCap* of the last bit of the *GoodCRC* Message in response to the *EPR_Mode* Message with the Action field set to 5 (*Exit*), *Sink* **Shall** initiate a *Hard Reset*.

6.4.10.3.2 Implicit Exit

EPR Mode **Shall** be exited as the side-effect of the *Power Role Swap* and *Fast Role Swap* processes. This is because at the end of these processes *V_{BUS}* will be at *vSafe5V* and the Ports will be in an *Implicit Contract*. The *New Source* will then send a *Source_Capabilities* Message (not an *EPR_Source_Capabilities* Message) to begin the process of

negotiating an SPR *Explicit Contract*. Once an SPR *Explicit Contract* is entered, the *Source* and *Sink* can then enter *EPR Mode* if needed.

6.4.10.3.3 Exits due to errors

Other critical errors can occur while in *EPR Mode*; these errors **shall** result in *Hard Reset* being initiated by the *Port* that detects the error. Some of these errors include:

- An **EPR_Mode** Message with the Action field set to 5 (**Exit**) to exit *EPR Mode* is received by a *Port* in an *Explicit Contract* with an EPR (A)PDO.
- The *Sink* receives an **EPR_Source_Capabilities** Message with an EPR (A)PDO in any of the first seven object positions.
- The (A)PDO in the **EPR_Request** Message does not match the (A)PDO in the latest **EPR_Source_Capabilities** Message pointed to by the Object Position field in the *RDO*.
- The *Source* receives a **Request** Message.
- The *Sink* receives a **Source_Capabilities** Message not in response to a **Get_Source_Cap** Message.

6.4.11 Source_Info Message

The **Source_Info** Message **Shall** be sent in response to a **Get_Source_Info** Message. The **Source_Info** Message contains one Source Information Data Object (SIDO).

The **Source_Info** Message returns a SIDO whose format **Shall** be as shown in [Figure 6.35, "Source_Info Message"](#) and [Table 6.51, "Source_Info Data Object \(SIDO\)"](#). The **Number of Data Objects** field in the **Source_Info** Message **Shall** be set to 1.

The **Port Maximum PDP**, **Port Present PDP**, **Port Reported PDP** and the **Port Type** are used to identify *Capabilities* of a *Source Port*.

Figure 6.35 Source_Info Message

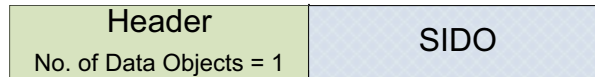


Table 6.51 Source_Info Data Object (SIDO)

Bit(s)	Field	Description
B31	Port Type	<ul style="list-style-type: none">0 = <i>Managed Capability Port</i>1 = <i>Guaranteed Capability Port</i>
B30...24	Reserved	Shall be set to zero
B23...16	Port Maximum PDP	Power the <i>Port</i> is designed to supply
B15...8	Port Present PDP	Power the <i>Port</i> is presently capable of supplying
B7...0	Port Reported PDP	Power the <i>Port</i> is actually advertising

6.4.11.1 Port Type Field

Port Type is a **Static** field that **Shall** be used to indicate whether the amount of power the *Port* can provide is fixed or can change dynamically.

For *Ports* that are part of a *Shared Capacity Group*, the **Port Type** field **Shall** be set to *Managed Capability Port*.

For *Ports* not part of a *Shared Capacity Group*, the **Port Type** field **May** be set to either *Managed Capability Port* or *Guaranteed Capability Port*.

6.4.11.2 Port Maximum PDP Field

Port Maximum PDP is a **Static** field that **Shall** report the integer portion of the *PDP Rating* of the *Port*. A *Guaranteed Capability Port* (as indicated by the **Port Type** field being set to '1') **Shall** always be capable of supplying this amount of power. A *Managed Capability Port* (as indicated by the **Port Type** field being set to '0') **Shall** be able to offer this amount of power at some time.

The **Port Maximum PDP** **Shall** be the same as the larger of the *SPR Source PDP Rating* and the *EPR Source PDP Rating* in the **Source_Capabilities_Extended** Message.

6.4.11.3 Port Present PDP Field

The **Port Present PDP** field **Shall** indicate the integer part of the amount of power the *Port* is presently capable of supplying including limitations due to *Cable Capabilities* or abnormal operating conditions (e.g., elevated temperature, low input voltage, etc.).

A *Guaranteed Capability Port* **Shall** always set its **Port Present PDP** to be the same as its **Port Maximum PDP** or the highest possible value when limited.

A *Managed Capability Port* that is part of a *Shared Capacity Group* **shall** set its **Port Present PDP** to Shared Port Power Available as defined in [USB Type-C 2.4] or to a lower value when limited.

A *Managed Capability Port* that is part of an *Assured Capacity Group* **shall** set its **Port Present PDP** to the **Port Maximum PDP** or the highest value possible when limited.

6.4.11.4 Port Reported PDP Field

The **Port Reported PDP** field **shall** track the amount of power the *Port* is offering in its **Source_Capabilities** Message or **EPR_Source_Capabilities** Message. The **Port Reported PDP** field **may** be dynamic or **Static** depending on the *Port*'s other characteristics such as Managed/Guaranteed Capability, SPR/EPR Mode, its power policy etc.

Note: The **Port Reported PDP** field is computed as the integer part of, the largest of the products of the voltage times current of the *Fixed Supply PDOs* returned in the **Source_Capabilities** Message or **EPR_Source_Capabilities** Messages.

6.4.12 Revision Message

The **Revision** Message **Shall** be sent in response to the **Get_Revision** Message sent by the *Port Partner*. This Message is used to identify the highest *Revision* the *Port* is capable of operating at. The **Revision** Message contains one **Revision** Message Data Object (RMDO).

The **Revision** Message returns an RMDO whose format **Shall** be as shown in [Figure 6.36, "Revision Message Data Object"](#) and [Table 6.52, "Revision Message Data Object \(RMDO\)"](#). The **Number of Data Objects** field in the **Revision** Message **Shall** be set to 1.

Figure 6.36 Revision Message Data Object

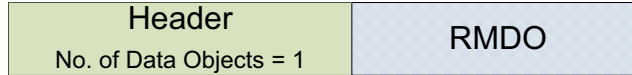


Table 6.52 Revision Message Data Object (RMDO)

Bit(s)	Description
B31...28	Revision.major
B27...24	Revision.minor
B23...20	Version.major
B19...16	Version.minor
B15...0	Reserved, Shall be set to zero.

E.g., for Revision 3.2, Version 1.1 the fields would be the following:

- Revision.major = 0011b
- Revision.minor = 0010b
- Version.major = 0001b
- Version.minor = 0001b

6.5 Extended Message

An *Extended Message* **Shall** contain an *Extended Message Header* (indicated by the **Extended** field in the *Message Header* being set) and be followed by zero or more data bytes. Additional bytes that might be added to existing *Messages* in future *Revision* of this specification **Shall** be **Ignored**.

The format of the *Extended Message* is defined by the *Message Header*'s **Message Type** field and is summarized in [Table 6.53, "Extended Message Types"](#). The Sent by column indicates entities which **May** send the given *Message* (*Source*, *Sink* or *Cable Plug*); entities not listed **Shall Not** issue the corresponding *Message*. The "Valid Start of Packet" column indicates the *Messages* which **Shall** only be issued in *SOP Packets* and the *Messages* which **May** be issued in *SOP* Packets*.

Table 6.53 Extended Message Types

Bits 4...0	Type	Sent by	Description	Valid Start of Packet
0 0000	Reserved		All values not explicitly defined are Reserved and Shall Not be used.	
0 0001	Source_Capabilities_Extended	Source or Dual-Role Power	See Section 6.5.1	SOP only
0 0010	Status	Source, Sink or Cable Plug	See Section 6.5.2	SOP*
0 0011	Get_Battery_Cap	Source or Sink	See Section 6.5.3	SOP only
0 0100	Get_Battery_Status	Source or Sink	See Section 6.5.4	
0 0101	Battery_Capabilities	Source or Sink	See Section 6.5.5	SOP only
0 0110	Get_Manufacturer_Info	Source or Sink	See Section 6.5.6	SOP*
0 0111	Manufacturer_Info	Source, Sink or Cable Plug	See Section 6.5.7	SOP*
0 1000	Security_Request	Source or Sink	See Section 6.5.8.1	SOP*
0 1001	Security_Response	Source, Sink or Cable Plug	See Section 6.5.8.2	SOP*
0 1010	Firmware_Update_Request	Source or Sink	See Section 6.5.9.1	SOP*
0 1011	Firmware_Update_Response	Source, Sink or Cable Plug	See Section 6.5.9.2	SOP*
0 1100	PPS_Status	Source	See Section 6.5.10	SOP only
0 1101	Country_Info	Source or Sink	See Section 6.5.12	SOP only
0 1110	Country_Codes	Source or Sink	See Section 6.5.11	SOP only
0 1111	Sink_Capabilities_Extended	Sink or Dual-Role Power	See Section 6.5.13	SOP only
1 0000	Extended_Control	Source or Sink	See Section 6.5.14	SOP only
1 0001	EPR_Source_Capabilities	Source or Dual-Role Power	See Section 6.5.15.2	SOP only
1 0010	EPR_Sink_Capabilities	Sink or Dual-Role Power	See Section 6.5.15.3	SOP only
1 0011... 1 1101	Reserved		All values not explicitly defined are Reserved and Shall Not be used.	
1 1110	Vendor_Defined_Extended	Source, Sink or Cable Plug	See Section 6.5.16	SOP*
1 1111	Reserved		All values not explicitly defined are Reserved and Shall Not be used.	

6.5.1 Source_Capabilities_Extended Message

The **Source_Capabilities_Extended** Message **Should** be sent in response to a **Get_Source_Cap_Extended** Message. The **Source_Capabilities_Extended** Message enables a *Source* or a *DRP* to inform the *Sink* about its *Capabilities* as a *Source*.

The **Source_Capabilities_Extended** Message **Shall** return a 25-byte Source Capabilities Extended Data Block (SCEDB) whose format **Shall** be as shown in [Figure 6.37, "Source_Capabilities_Extended Message"](#) and [Table 6.54, "Source Capabilities Extended Data Block \(SCEDB\)"](#).

Figure 6.37 Source_Capabilities_Extended Message

Extended Header	SCEDB
Data Size = 25	(25-byte Data Block)

Table 6.54 Source Capabilities Extended Data Block (SCEDB)

Offset	Field	Description	
0	VID	Vendor ID (assigned by the USB-IF)	
2	PID	Product ID (assigned by the manufacturer)	
4	XID	Value provided by the USB-IF assigned to the product	
8	FW Version	Firmware version number	
9	HW Version	Hardware version number	
10	Voltage Regulation	Bit	Description
		1...0	<ul style="list-style-type: none">00b: 150mA/μs Load Step (default)01b: 500mA/μs Load Step11b...10b: Reserved and Shall Not be used.
		2	<ul style="list-style-type: none">0b: 25% <i>IoC</i> (default)1b: 90% <i>IoC</i>
		3...7	Reserved and Shall Not be used
11	Holdup Time	Output will stay with regulated limits for this number of milliseconds after removal of the AC from the input. <ul style="list-style-type: none">0x00 = feature not supported Note: A value of at least 3ms Should be used (see Section 7.1.12.2, "Holdup Time Field").	
12	Compliance	Compliance in <i>SPR Mode</i> :	
		Bit	Description
		0	<i>LPS</i> compliant when set
		1	PS1 compliant when set
		2	PS2 compliant when set
		3...7	Reserved and Shall Not be used
13	Touch Current	Bit	Description
		0	Low touch current EPS when set
		1	Ground pin supported when set
		2	Ground pin intended for protective earth when set
		3...7	Reserved and Shall Not be used

Table 6.54 Source Capabilities Extended Data Block (SCEDB) (Continued)

Offset	Field	Description	
14	<i>Peak Current1</i>	Bit	Description
		0...4	Percent overload in 10% increments Values higher than 25 (11001b) are clipped to 250%.
		5...10	Overload period in 20ms
		11...14	Duty cycle in 5% increments
		15	<i>V_{BUS}</i> voltage droop
16	<i>Peak Current2</i>	Bit	Description
		0...4	Percent overload in 10% increments Values higher than 25 (11001b) are clipped to 250%.
		5...10	Overload period in 20ms
		11...14	Duty cycle in 5% increments
		15	<i>V_{BUS}</i> voltage droop
18	<i>Peak Current3</i>	Bit	Description
		0...4	Percent overload in 10% increments Values higher than 25 (11001b) are clipped to 250%.
		5...10	Overload period in 20ms
		11...14	Duty cycle in 5% increments
		15	<i>V_{BUS}</i> voltage droop
20	<i>Touch Temp</i>	Temperature conforms to: <ul style="list-style-type: none"> 0 = <i>[IEC 60950-1]</i> (default) 1 = <i>[IEC 62368-1]</i> TS1 2 = <i>[IEC 62368-1]</i> TS2 Note: All other values Reserved and Shall Not be used.	
21	<i>Source Inputs</i>	Bit	Description
		0	<ul style="list-style-type: none"> 0b: No external supply 1b: External supply present
		1	If bit 0 is set: <ul style="list-style-type: none"> 0b: External supply is constrained. 1b: External supply is unconstrained. If bit 0 is not set Reserved and Shall be set to zero
		2	<ul style="list-style-type: none"> 0b: No internal <i>Battery</i> 1b: Internal <i>Battery</i> present
		3...7	Reserved and Shall be set to zero
22	<i>Number of Batteries/ Battery Slots</i>	Upper Nibble = Number of Hot Swappable <i>Battery Slots</i> (0...4) Lower Nibble = Number of <i>Fixed Batteries</i> (0...4)	
23	<i>SPR Source PDP Rating</i>	0...6: <i>Source PDP Rating</i> (<i>EPR Source's PDP Rating</i> when operating in <i>SPR Mode</i>). 7: Reserved and Shall be set to zero	
24	<i>EPR Source PDP Rating</i>	0...7: <i>EPR Source PDP Rating</i>	

6.5.1.1 Vendor ID (VID) Field

The **VID** field **Shall** contain the 16-bit *Vendor ID (VID)* assigned to the *Source's* vendor by the USB-IF. If the vendor does not have a *VID*, the **VID** field **Shall** be set to 0xFFFF. *Devices* that have a USB data interface **Shall** report the same *VID* as the *idVendor* in the Standard Device Descriptor (see *[USB 2.0]* and *[USB 3.2]*).

6.5.1.2 Product ID (PID) Field

The **PID** field **shall** contain the 16-bit Product ID (PID) assigned by the *Source's* vendor. *Devices* that have a USB data interface **shall** report the same PID as the idProduct in the Standard Device Descriptor (see [\[USB 2.0\]](#) and [\[USB 3.2\]](#)).

6.5.1.3 XID Field

The **XID** field **shall** contain the 32-bit XID provided by the USB-IF to the vendor who in turns assigns it to a product. If the vendor does not have an XID, then it **shall** return zero in this field (see [\[USB 2.0\]](#) and [\[USB 3.2\]](#)).

6.5.1.4 Firmware Version Field

The **FW Version** field **shall** contain an 8-bit firmware version number assigned to the device by the vendor.

6.5.1.5 Hardware Version Field

The **HW Version** field **shall** contain an 8-bit hardware version number assigned to the device by the vendor.

6.5.1.6 Voltage Regulation Field

The **Voltage Regulation** field contains bits covering Load Step Slew Rate and Magnitude.

See [Section 7.1.12.1, "Voltage Regulation Field"](#) for further details.

6.5.1.6.1 Load Step Slew Rate

The *Source* **shall** report its load step response capability in bits 0...1 of the **Voltage Regulation** bit field.

6.5.1.6.2 Load Step Magnitude

The *Source* **shall** report its load step magnitude rate as a percentage of *IoC* in bit 2 of the **Voltage Regulation** field.

6.5.1.7 Holdup Time Field

The **Holdup Time** field **shall** contain the *Source's* holdup time (see [Section 7.1.12.2, "Holdup Time Field"](#)).

6.5.1.8 Compliance Field

The **Compliance** is **Static** and **shall** contain the standards the *Source* is compliant with in SPR (see [Section 7.1.12.3, "Compliance Field"](#)).

6.5.1.9 Touch Current Field

The **Touch Current** field reports whether the *Source* meets certain leakage current levels and if it has a ground pin.

A *Source* **shall** set the **Touch Current** bit (bit 0) when their leakage current is less than 65µA rms when *Source's* maximum capability is less than or equal to 30W, or when their leakage current is less than 100 µA rms when its power capability is between 30W and 100W. The total combined leakage current **shall** be measured in accordance with [\[IEC 60950-1\]](#) when tested at 250VAC rms at 50 Hz.

A *Source* with a ground pin **shall** set the Ground pin bit (bit 1).

A *Source* whose Ground pin is intended to be connected to a protective earth **shall** set both bit1 and bit 2.

6.5.1.10 Peak Current Field

The **Peak Current1**/**Peak Current2**/**Peak Current3** fields **shall** contain the combinations of Peak Current that the *Source* supports (see [Section 7.1.12.4, "Peak Current"](#)).

Peak Current provides a means for *Source* report its ability to provide current in excess of the *Negotiated* amount for short periods. The Peak Current descriptor defines up to three combinations of % overload, duration and duty

cycle defined as **Peak Current1**, **Peak Current2** and **Peak Current3** that the *Source* supports. A *Source* **May** offer no Peak Current capability. A *Source* **Shall** populate unused Peak Current bit fields with zero.

The Bit Fields within **Peak Current1**, **Peak Current2** and **Peak Current3** contain the following subfields:

- Percentage Overload
 - **Shall** be the maximum peak current reported in 10% increments as a percentage of the *Negotiated* operating current (*IoC*) offered by the *Source*. Values higher than 25 (11001b) are clipped to 250%.
- Overload Period
 - **Shall** be the minimum rolling average time window in 20ms increments, where a value of 20ms is recommended.
- Duty Cycle
 - **Shall** be the maximum percentage of overload period reported in 5% increments. The values **Should** be 5%, 10% and 50% for PeakCurrent1, PeakCurrent2, and PeakCurrent3, respectively.
- *VBUS* Droop
 - **Shall** be set to one to indicate there is an additional 5% voltage droop on *VBUS* when the overload conditions occur as defined by **vSrcPeak**. However, it is recommended that the *Source* **Should** provide *VBUS* in the range of **vSrcNew** when overload conditions occur and set this bit to zero.

6.5.1.11 Touch Temp Field

The **Touch Temp** field **Shall** report the IEC standard used to determine the surface temperature of the *Source*'s enclosure. Safety limits for the *Source*'s touch temperature are set in applicable product safety standards (e.g., **[IEC 60950-1]** or **[IEC 62368-1]**). The *Source* **May** report when its touch temperature performance conforms to the TS1 or TS2 limits described in **[IEC 62368-1]**.

6.5.1.12 Source Inputs Field

The **Source Inputs** field **Shall** identify the possible inputs that provide power to the *Source*:

- When bit 0 is set, the *Source* can be sourced by an external power supply.
- When bits 0 and 1 are set, the *Source* can be sourced by an external power supply which is assumed to be effectively "infinite" i.e., it won't run down over time.
- When bit 2 is set the *Source* can be sourced by an internal *Battery*.

Note: Some *Sources* are only powered by a *Battery* (e.g., an automobile) rather than the more common AC Supply.

Note: Bit 2 **May** be set independently of bits 0 and 1.

6.5.1.13 Number of Batteries/Battery Slots Field

The **Number of Batteries/Battery Slots** field **Shall** report the number of *Fixed Batteries* and Hot Swappable *Battery Slots* the *Source* supports. This field **Shall** independently report the number of *Battery Slots* and the number of *Fixed Batteries*.

A *Source* **Shall** have no more than 4 *Fixed Batteries* and no more than 4 *Battery Slots*.

Fixed Batteries **Shall** be numbered consecutively from 0 to 3. The number assigned to a given *Fixed Battery* **Shall** **Not** change between *Attach* and *Detach*.

Battery Slots **Shall** be numbered consecutively from 4 to 7. The number assigned to a given *Battery Slot* **Shall** **Not** change between *Attach* and *Detach*.

6.5.1.14 SPR Source PDP Rating Field

For an *SPR Source* the *SPR Source PDP Rating* field **Shall** report the integer portion of the *PDP Rating* of the *Port*.

For an *EPR Source*, the *SPR Source PDP Rating* field **Shall** report the integer portion of the maximum amount of power that the *Port* is designed to deliver in SPR Mode.

The *SPR Source PDP Rating* field that is reported **Shall** be **Static**.

6.5.1.15 EPR Source PDP Rating Field

For an *EPR Source* the *EPR Source PDP Rating* field **Shall** report the integer portion of the *PDP Rating* of the *Port*.

For an *SPR Source* this field **Shall** be set to zero.

The *EPR Source PDP Rating* field that is reported **Shall** be **Static**.

6.5.2 Status Message

The **Status Message** *Shall* be sent in response to a **Get_Status Message**. The content of the **Status Message** depends on the target of the **Get_Status Message**. When sent to *SOP* the **Status Message** returns the status of the *Port's Port Partner*. When sent to *SOP'* or *SOP''* the **Status Message** returns the status of one of the *Active Cable's Cable Plugs*.

6.5.2.1 SOP Status Message

A **Status Message**, sent in response to **Get_Status Message** to *SOP*, enables a *Port* to inform its *Port Partner* about the present status of the *Source* or *Sink*. Typically, a **Get_Status Message** will be sent by the *Port* after receipt of an **Alert Message**. Some of the reported events are critical such as *OCP*, *OVP* and *OTP*, while others are informational such as change in a *Battery's* status from charging to neither charging nor discharging.

The **Status Message** returns a 7-byte Status Data Block (SDB) whose format *Shall* be as shown in [Figure 6.38, "SOP Status Message"](#) and [Table 6.55, "SOP Status Data Block \(SDB\)"](#).

Figure 6.38 SOP Status Message

Extended Header	SDB
Data Size = 7	(7-byte block)

Table 6.55 SOP Status Data Block (SDB)

Offset (Byte)	Field	Description	
0	<i>Internal Temp</i>		<i>Source</i> or <i>Sink's</i> internal temperature in °C <ul style="list-style-type: none">0 = feature not supported1 = temperature is less than 2°C.2-255 = temperature in °C.
1	<i>Present Input</i>	Bit	Description
		0	Reserved and Shall be set to zero
		1	External Power when set
		2	External Power AC/DC (Valid when Bit 1 set) <ul style="list-style-type: none">0: DC1: AC Reserved when Bit 1 is zero
		3	Internal Power from <i>Battery</i> when set
		4	Internal Power from non- <i>Battery</i> power source when set
2	<i>Present Battery Input</i>	5...7	Reserved and Shall be set to zero
		When <i>Present Input</i> field bit 3 set Shall contain the bit corresponding to the <i>Battery</i> or <i>Batteries</i> providing power: <ul style="list-style-type: none">Upper nibble = Hot Swappable Battery (b7...4)Lower nibble = <i>Fixed Battery</i> (b3...0) When <i>Present Input</i> field bit 3 is not set this field is Reserved and Shall be set to zero.	

Table 6.55 SOP Status Data Block (SDB) (Continued)

Offset (Byte)	Field	Description		
3	<i>Event Flags</i>	Bit	Flag	Description
		0		Reserved and Shall be set to zero
		1	<i>OCP Event</i>	<i>OCP</i> event when set
		2	<i>OTP Event</i>	<i>OTP</i> event when set
		3	<i>OVP Event</i>	<i>OVP</i> event when set
		4	<i>CL/CV Mode</i>	In <i>PPS Mode</i> only: <i>CL</i> mode when set, <i>CV</i> mode when cleared
		5...7		Reserved and Shall be set to zero
4	<i>Temperature Status</i>	Bit	Description	
		0	Reserved and Shall be set to zero	
		1...2	<ul style="list-style-type: none"> 00 – Not Supported. 01 – Normal 10 – Warning 11 – Over temperature 	
		3...7	Reserved and Shall be set to zero	
5	<i>Power Status</i>	Bit	Description	
		0	Reserved and Shall be set to zero	
		1	Source power limited due to cable supported current	
		2	Source power limited due to insufficient power available while sourcing other ports	
		3	Source power limited due to insufficient external power	
		4	Source power limited due to Event Flags in place (Event Flags must also be set)	
		5	Source power limited due to temperature	
		6...7	Reserved and Shall be set to zero	

Table 6.55 SOP Status Data Block (SDB) (Continued)

Offset (Byte)	Field	Description	
6	<i>Power State Change</i>	Bit	Description
		0...2	<i>New Power State</i>
			Value Description
			0 Status not supported
			1 S0
			2 Modern Standby
			3 S3
			4 S4
			5 S5 (Off with battery, wake events supported)
			6 G3 (Off with no battery, wake events not supported)
			7 Reserved and Shall be set to zero
		3...5	<i>New Power State indicator</i>
			Value Description
			0 Off LED
			1 On LED
			2 Blinking LED
			3 Breathing LED
		4...7	Reserved and Shall be set to zero
		6...7	Reserved and Shall be set to zero

6.5.2.1.1 Internal Temp Field

The *Internal Temp* field reports the instantaneous temperature of a portion of the *Source* or *Sink*.

6.5.2.1.2 Present Input Field

The *Present Input* field indicates which supplies are presently powering the *Source* or *Sink*.

The following bits are defined:

- Bit 1: indicates that an external power source is present.
- Bit 2: indicates whether the external unconstrained power source is AC or DC.
- Bit 3: indicates that power is being provided from *Battery*.
- Bit4: indicates an alternative internal source of power that is not a *Battery*.

6.5.2.1.3 Present Battery Input Field

The *Present Battery Input* field indicates which *Battery* or Batteries are presently supplying power to the *Source* or *Sink*. The *Present Battery Input* field is only **Valid** when the *Present Input* field indicates that there is Internal Power from *Battery*.

The upper nibble of the field indicates which Hot Swappable *Battery*/Batteries are supplying power with bit 4 in upper nibble corresponding to *Battery* 4 and bit 7 in the upper nibble corresponding to *Battery* 7 (see [Section 6.5.3, "Get Battery Cap Message"](#) and [Section 6.5.4, "Get Battery Status Message"](#)).

The lower nibble of the field indicates which *Fixed Battery*/Batteries are supplying power with bit 0 in lower nibble corresponding to *Battery* 0 and bit 3 in the lower nibble corresponding to *Battery* 3 (see [Section 6.5.3, "Get Battery Cap Message"](#) and [Section 6.5.4, "Get Battery Status Message"](#)).

6.5.2.1.4 Event Flags Field

The **Event Flags** field returns event flags. The *OTP*, *OVP* and *OCF* event flags **Shall** be set when there is an event and **Shall** only be cleared when read with the **Get_Status** Message.

When the **OTP Event** flag is set the **Temperature Status** field **Shall** also be set to over temperature.

The **CL/CV Mode** flag is only **Valid** when operating as a Programmable Power Supply and **Shall** be **Ignored** otherwise. When the *Source* is operating as a Programmable Power Supply the **CL/CV Mode** flag **Shall** be set when operating in **Current Limit** mode (**CL**) and **Shall** be cleared when operating in **Constant Voltage** mode (**CV**).

6.5.2.1.5 Temperature Status Field

The **Temperature Status** field returns the current temperature status of the device either: normal, warning or over temperature. When the **Temperature Status** field is set to over temperature the **OTP Event** flag **Shall** also be set.

6.5.2.1.6 Power Status Field

The **Power Status** field indicates the current status of a *Source*. A non-zero return of the field indicates *Advertised Source* power is being reduced for either:

- The cable does not support the full *Source* current.
- The *Source* is supplying power to other ports and is unable to provide its full power.
- The external power to the *Source* is insufficient to support full power.
- An Event has occurred that is causing the *Source* to reduce its *Advertised* power.

A *Sink* **Shall** set this field to zero.

6.5.2.1.7 Power state change

The **Power State Change** field contains two status bytes; the *New Power State* and *New Power State indicator* status bytes.

6.5.2.1.7.1 New power state

The *New Power State* status byte indicates a power state change to one of the specified power states. Any device that supports the ACPI standard system power states **shall** use the ACPI states. For devices that do not support the ACPI power states, the following mapping **should** be used:

- High power (on) state -> S0
- Sleep state -> S3
- Low power (off) state -> S5 or G3

6.5.2.1.7.2 New power state indicator

The **New Power State indicator** status byte defines the host's desired indicator for the specified power state. This indicator allows several possibilities for predefined behaviors that the host can specify to indicate its system power state to the user via the downstream device. The **New Power State indicator** is a "best effort" indicator. If the device cannot provide the requested indicator, then it provides the best indicator that it can. If a Breathing indicator cannot be provided, then a Blinking indicator **should** be provided. If a Blinking indicator cannot be provided, then a constant on indicator **should** be provided.

New Power State indicators in decreasing precedence:

- Breathing
- Blinking
- Constant on
- No indicator

6.5.2.2 SOP'/SOP'' Status Message

A **Status Message**, sent in response to a **Get_Status Message** to *SOP'* or *SOP''*, enables a *Source* or *Sink* to get the present status of the Cable's *Cable Plug(s)*. Typically, a **Get_Status Message** will be used by the *USB Host* and/or *USB Device* to manage the Cable's *Cable Plug(s)* temperature. The **Status Message** returns a 2-byte Status Data Block (SDB) whose format **Shall** be as shown in [Figure 6.39, "SOP'/SOP'' Status Message"](#) and [Table 6.56, "'SOP'/SOP'' Status Data Block \(SPDB\)'"](#).

Passive *Cable Plugs* **Shall Not** indicate Thermal Shutdown.

Figure 6.39 SOP'/SOP'' Status Message

Extended Header Data Size = 2	SPDB (2-byte block)
----------------------------------	------------------------

Table 6.56 "SOP'/SOP'' Status Data Block (SPDB)"

Offset (Byte)	Field	Value	Description	
0	Internal Temp	Unsigned Int	<i>Cable Plug's</i> internal temperature in °C. <ul style="list-style-type: none">0 = feature not supported1 = temperature is less than 2°C.2...255 = temperature in °C.	
1	Flags	Bit Field	Bit	Description
			0	Thermal Shutdown
			1...7	Reserved and Shall be set to zero

6.5.2.2.1 Internal Temp Field

The **Internal Temp** field reports the instantaneous temperature of the plug in °C. The internal temperature **Shall** be monotonic. The *Cable Plug* **Shall** report its internal temperature every **tACTempUpdate**.

6.5.2.2.2 Thermal Shutdown Field

The **Flags** flag **Shall** also be set when the plug's internal temperature exceeds the Internal Maximum Temperature reported in the *Active Cable* VDO. Once this bit has been set, it **Shall** remain set and the plug **Shall** remain in Thermal Shutdown until there is a *Hard Reset* or the *Active Cable's* power is removed. The Thermal Shutdown flag **Shall Not** be cleared by a *Cable Reset*.

6.5.3 Get_Battery_Cap Message

The **Get_Battery_Cap** (Get Battery Capabilities) Message is used to request the capability of a *Battery* present in its *Port Partner*. The *Port* **Shall** respond by returning a **Battery_Capabilities** Message (see [Section 6.5.5, "Battery_Capabilities Message"](#)) containing a Battery Capabilities Data Block (BCDB) for the targeted *Battery*.

The **Get_Battery_Cap** Message contains a 1-byte Get Battery Cap Data Block (GBCDB), whose format **Shall** be as shown in [Figure 6.40, "Get_Battery_Cap Message"](#) and [Table 6.57, "Get Battery Cap Data Block \(GBCDB\)"](#). This block defines for which *Battery* the request is being made.

The **Data Size** field in the **Get_Battery_Cap** Message **Shall** be set to 1.

Figure 6.40 Get_Battery_Cap Message

Extended Header	GBCDB
Data Size = 1	

Table 6.57 Get Battery Cap Data Block (GBCDB)

Offset	Field	Description
0	Battery Cap Ref	Number of the <i>Battery</i> indexed from zero: <ul style="list-style-type: none">Values 0...3 represent the <i>Fixed Batteries</i>.Values 4...7 represent the Hot Swappable Batteries.Values 8...255 are Reserved and Shall Not be used.

6.5.4 Get_Battery_Status Message

The **Get_Battery_Status** (Get Battery Status) Message is used to request the status of a *Battery* present in its *Port Partner*. The *Port* **Shall** respond by returning a **Battery_Status** Message (see [Section 6.4.5, "Battery_Status Message"](#)) containing a Battery Status Data Object (BSDO) for the targeted *Battery*.

The **Get_Battery_Status** Message contains a 1-byte Get Battery Status Data Block (GBSDB) whose format **Shall** be as shown in [Figure 6.41, "Get_Battery_Status Message"](#) and [Table 6.58, "Get Battery Status Data Block \(GBSDB\)"](#). This block contains details of the requested *Battery*. The **Data Size** field in the **Get_Battery_Status** Message **Shall** be set to 1.

Figure 6.41 Get_Battery_Status Message

Extended Header	GBSDB
Data Size = 1	

Table 6.58 Get Battery Status Data Block (GBSDB)

Offset	Field	Description
0	Battery Status Ref	Number of the <i>Battery</i> indexed from zero: <ul style="list-style-type: none">Values 0...3 represent the <i>Fixed Batteries</i>.Values 4...7 represent the Hot Swappable Batteries.Values 8...255 are Reserved and Shall Not be used.

6.5.5 Battery_Capabilities Message

The **Battery_Capabilities** Message is sent in response to a **Get_Battery_Cap** Message. The **Battery_Capabilities** Message contains one Battery Capability Data Block (BCDB) for one of the Batteries its supports as reported by **Number of Batteries/Battery Slots** field in the **Source_Capabilities_Extended** Message. The returned BCDB **Shall** correspond to the **Battery** requested in the **Battery Cap Ref** field contained in the **Get_Battery_Cap** Message.

The **Battery_Capabilities** Message returns a 9-byte BCDB whose format **Shall** be as shown in [Figure 6.42](#), "[Battery_Capabilities Message](#)" and [Table 6.59](#), "[Battery Capability Data Block \(BCDB\)](#)".

Figure 6.42 Battery_Capabilities Message

Extended Header Data Size = 9	BCDB
----------------------------------	------

Table 6.59 Battery Capability Data Block (BCDB)"

Offset (Byte)	Field	Description		
0	VID	Vendor ID (assigned by the USB-IF)		
2	PID	Product ID (assigned by the manufacturer)		
4	Battery Design Capacity	Battery's design capacity in 0.1 WH Note: <ul style="list-style-type: none">0x0000 = Battery not present0xFFFF = design capacity unknown		
6	Battery Last Full Charge Capacity	Battery's last full charge capacity in 0.1 WH Note: <ul style="list-style-type: none">0x0000 = Battery not present0xFFFF = last full charge capacity unknown		
8	Battery Type	Bit	Field	Description
		0	Invalid Battery Reference	Invalid Battery reference when set.
		1...7	---	Reserved

6.5.5.1 Vendor ID (VID)

The **VID** field **Shall** contain the manufacturer **VID** associated with the **Battery**, as assigned by the USB-IF, or 0xFFFF in the case that no such **VID** exists.

If the **Battery Cap Ref** field in the **Get_Battery_Cap** Message is **Invalid**, the **VID** field **Shall** be 0xFFFF.

6.5.5.2 Product ID (PID)

The following rules apply to the **PID** field. When the **VID**:

- Belongs to the *Battery* vendor the **PID** field **Shall** contain the *Battery*'s 16-bit product identifier designated by the *Battery* vendor.
- Belongs to the *Device* vendor the **PID** field **Shall** contain the *Battery*'s 16-bit product identifier designated by the *Device* vendor.
- Is 0xFFFF the **PID** field **Shall** be set to 0x0000.

6.5.5.3 Battery Design Capacity Field

The **Battery Design Capacity** field **Shall** return the *Battery*'s design capacity in tenths of WH. If the *Battery* is Hot Swappable and is not present, the **Battery Design Capacity** field **Shall** be set to zero. If the *Battery* is unable to report its Design Capacity, the **Battery Design Capacity** field **Shall** be set to 0xFFFF.

6.5.5.4 Battery Last Full Charge Capacity Field

The **Battery Last Full Charge Capacity** field **Shall** contain the *Battery*'s last full charge capacity in tenths of WH. If the *Battery* is Hot Swappable and is not present, the **Battery Last Full Charge Capacity** field **Shall** be set to zero. If the *Battery* is unable to report its Design Capacity, the **Battery Last Full Charge Capacity** field **Shall** be set to 0xFFFF.

6.5.5.5 Battery Type Field

The **Battery Type** field is used to report additional information about the *Battery*'s *Capabilities*.

6.5.5.5.1 Invalid Battery Reference

The **Invalid Battery Reference** bit **Shall** be set when the **Get_Battery_Cap** Message contains a reference to a *Battery* that does not exist.

6.5.6 Get_Manufacturer_Info Message

The **Get_Manufacturer_Info** (Get Manufacturer Info) Message is sent by a *Port* to request manufacturer specific information relating to its *Port Partner*, *Cable Plug* or of a *Battery* behind a *Port*. The *Port* **shall** respond by returning a **Manufacturer_Info** Message ([Section 6.5.7, "Manufacturer_Info Message"](#)) containing a Manufacturer Info Data Block (MIDB). Support for this feature by the *Cable Plug* is **Optional Normative**.

The **Get_Manufacturer_Info** Message contains a 2-byte Get Manufacturer Info Data Block (GMIDB). This block defines whether it is the *Device* or *Battery* manufacturer information being requested and for which *Battery* the request is being made.

The **Get_Manufacturer_Info** Message returns a GMIDB whose format **shall** be as shown in [Figure 6.43, "Get_Manufacturer_Info Message"](#) and [Table 6.60, "Get Manufacturer Info Data Block \(GMIDB\)"](#).

Figure 6.43 Get_Manufacturer_Info Message

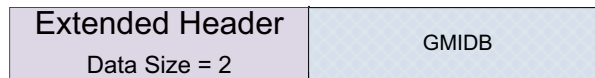


Table 6.60 Get Manufacturer Info Data Block (GMIDB)

Offset	Field	Description
0	<i>Manufacturer Info Target</i>	<ul style="list-style-type: none">0: <i>Port/Cable Plug</i>1: <i>Battery</i>255...2: Reserved and shall Not be used.
1	<i>Manufacturer Info Ref</i>	<p>If the <i>Manufacturer Info Target</i> field is <i>Battery</i> (01b) the <i>Manufacturer Info Ref</i> field shall contain the <i>Battery</i> number reference which is the number of the <i>Battery</i> indexed from zero:</p> <ul style="list-style-type: none">Values 0...3 represent the <i>Fixed Batteries</i>.Values 4...7 represent the <i>Hot Swappable Batteries</i>. <p>Otherwise, this field is Reserved and shall be set to zero.</p>

6.5.7 Manufacturer_Info Message

The **Manufacturer_Info** Message **Shall** be sent in response to a **Get_Manufacturer_Info** Message. The **Manufacturer_Info** Message contains the USB **VID** and the Vendor's **PID** to identify the device or **Battery** and the device or **Battery**'s manufacturer byte array in a variable length Data Block of up to **MaxExtendedMsgLegacyLen**.

The **Manufacturer_Info** Message returns a Manufacturer Info Data Block (MIDB) whose format **Shall** be as shown in [Figure 6.44, "Manufacturer_Info Message"](#) and [Table 6.61, "Manufacturer Info Data Block \(MIDB\)"](#).

Figure 6.44 Manufacturer_Info Message

Extended Header Data Size = 5..26	MIDB
--------------------------------------	------

Table 6.61 Manufacturer Info Data Block (MIDB)

Offset	Field	Description
0	VID	Vendor ID (assigned by the USB-IF)
2	PID	Product ID (assigned by the manufacturer)
4	Manufacturer String	Vendor defined null terminated string of 0...21 characters. If the Manufacturer Info Target field or Manufacturer Info Ref field in the Get_Manufacturer_Info Message is unrecognized the field Shall return a null terminated ASCII text string "Not Supported".

6.5.7.1 Vendor ID (VID)

If the in **Manufacturer Info Target** field in the **Get_Manufacturer_Info** Message is associated with this **Port/Cable Plug**, the **VID** field **Shall** contain:

- The manufacturer's **VID** associated with the **Port/Cable Plug**, as defined by the USB-IF, or
- 0xFFFF in the case that the vendor does not have a **VID**.

If the in **Manufacturer Info Target** field in the **Get_Manufacturer_Info** Message is associated with a **Device** that has a USB data interface, the **Device** **Shall** report the same **VID** as the **idVendor** in the Standard Device Descriptor (see [\[USB 2.0\]](#) and [\[USB 3.2\]](#)).

If the in **Manufacturer Info Target** field in the **Get_Manufacturer_Info** Message is associated with a **Battery**, the **VID** field **Shall** contain:

- The manufacturer **VID** associated with the **Battery** specified, as defined by the USB-IF, or
- 0xFFFF in the case that the vendor does not have a **VID**.

If the in **Manufacturer Info Target** field in the **Get_Manufacturer_Info** Message:

- Is **Invalid**, this **VID** field **Shall** be 0xFFFF.
- Is **Battery** (01b) and the **Manufacturer Info Ref** field is **Invalid**, the **VID** field **Shall** be 0xFFFF.

6.5.7.2 Product ID (PID)

If the **VID** is 0xFFFF, the **PID** field **Shall** contain 0x0000.

Otherwise:

- If the in *Manufacturer Info Target* field in the *Get_Manufacturer_Info* Message is associated with this *Port/Cable Plug*, the PID field **Shall** contain the device's 16-bit product identifier designated by the device vendor.
- If the in *Manufacturer Info Target* field in the *Get_Manufacturer_Info* Message is associated with a *Battery*:
 - And the VID belongs to the *Battery* vendor, the *PID* field **Shall** contain the *Battery*'s 16-bit product identifier designated by the *Battery* vendor.
 - And the VID belongs to the *Device* vendor, the *PID* field **Shall** contain the *Battery*'s 16-bit product identifier designated by the *Device* vendor.

6.5.7.3 Manufacturer String

The *Manufacturer String* field **Shall** contain the device's or *Battery*'s manufacturer string as defined by the vendor.

If the *Manufacturer Info Target* field or *Manufacturer Info Ref* field in the *Get_Manufacturer_Info* Message is unrecognized the field **Shall** return a null terminated ASCII text string "Not Supported".

6.5.8 Security Messages

The authentication process between *Port Partners* or a *Port* and *Cable Plug* is fully described in [\[USBTypeCAuthentication 1.0\]](#). This specification describes two *Extended Messages* used by the authentication process when applied to PD.

In the authentication process described in [\[USBTypeCAuthentication 1.0\]](#) there are three basic exchanges that serve to:

- Get the *Port* or *Cable Plug*'s certificates.
- Get the *Port* or *Cable Plug*'s digest.
- Challenge the *Port Partner* or *Cable Plug*.

Certificates are used to convey information, attested to by a signer, which attests to the *Port Partner's* or *Cable Plug's* authenticity. The *Port's* or *Cable Plug's* certificates are needed when a *Port* encounters a *Port Partner* or *Cable Plug* it has not been *Attached* to before. To minimize calculations after the initial *Attachment*, a *Port* can also use a digest consisting of hashes of the certificates rather than the certificates themselves. Once the *Port* has the certificates and has calculated the hashes, it stores the hashes and uses the digest in future exchanges. After the *Port* gets the certificates or digest, it challenges its *Port Partner* or the *Cable Plug* to detect replay attacks.

For further details refer to [\[USBTypeCAuthentication 1.0\]](#).

6.5.8.1 Security_Request

The *Security_Request* Message is used by a *Port* to pass a security data structure to its *Port Partner* or a *Cable Plug*.

The *Security_Request* Message contains a Security Request Data Block (SRQDB) whose format **shall** be as shown in [Figure 6.45, "Security_Request Message"](#). The contents of the SRQDB and its use are defined in [\[USBTypeCAuthentication 1.0\]](#).

Figure 6.45 Security_Request Message



6.5.8.2 Security_Response

The *Security_Response* Message is used by a *Port* or *Cable Plug* to pass a security data structure to the *Port* that sent the *Security_Request* Message.

The *Security_Response* Message contains a Security Response Data Block (SRPDB) whose format **shall** be as shown in [Figure 6.46, "Security_Response Message"](#). The contents of the SRPDB and its use are defined in [\[USBTypeCAuthentication 1.0\]](#).

Figure 6.46 Security_Response Message



6.5.9 Firmware Update Messages

The firmware update process between *Port Partners* or a *Port* and *Cable Plug* is fully described in [\[USBPDFirmwareUpdate 1.0\]](#). This specification describes two *Extended Messages* used by the firmware update process when applied to PD.

6.5.9.1 Firmware_Update_Request

The **Firmware_Update_Request** Message is used by a *Port* to pass a firmware update data structure to its *Port Partner* or a *Cable Plug*.

The **Firmware_Update_Request** Message contains a Firmware Update Request Data Block (FRQDB) whose format **shall** be as shown in [Figure 6.47, "Firmware_Update_Request Message"](#). The contents of the FRQDB and its use are defined in [\[USBPDFirmwareUpdate 1.0\]](#).

Figure 6.47 Firmware_Update_Request Message



6.5.9.2 Firmware_Update_Response

The **Firmware_Update_Response** Message is used by a *Port* or *Cable Plug* to pass a firmware update data structure to the *Port* that sent the **Firmware_Update_Request** Message.

The **Firmware_Update_Response** Message contains a Firmware Update Response Data Block (FRPDB) whose format **shall** be as shown in [Figure 6.48, "Firmware_Update_Response Message"](#). The contents of the FRPDB and its use are defined in [\[USBPDFirmwareUpdate 1.0\]](#).

Figure 6.48 Firmware_Update_Response Message



6.5.10 PPS_Status Message

The **PPS_Status** Message **Shall** be sent in response to a **Get_PPS_Status** Message. The **PPS_Status** Message enables a *Sink* to query the *Source* to get additional information about its operational state. The **Get_PPS_Status** Message and the **PPS_Status** Message **Shall** only be supported when the **Alert** Message is also supported.

The **PPS_Status** Message **Shall** return a 4-byte PPS Status Data Block (PPSSDB) whose format **Shall** be as shown in [Figure 6.49, "PPS_Status Message"](#) and [Table 6.62, "PPS Status Data Block \(PPSSDB\)"](#).

Figure 6.49 PPS_Status Message

Extended Header Data Size = 4	PPSSDB (4-byte Data Block)
----------------------------------	-------------------------------

Table 6.62 PPS Status Data Block (PPSSDB)

Offset (Byte)	Field	Description	
0	Output Voltage	2	<i>Source's</i> output voltage in 20mV units. When set to 0xFFFF, the <i>Source</i> does not support this field.
2	Output Current	1	<i>Source's</i> output current in 50mA units. When set to 0xFF, the <i>Source</i> does not support this field.
3	Real Time Flags	Bit	Description
		0	Reserved and Shall be set to zero
		1...2	PTF <ul style="list-style-type: none">PTF: 00 – Not SupportedPTF: 01 – NormalPTF: 10 – WarningPTF: 11 – Over temperature
			OMF OMF (Operating Mode Flag) is set when operating in Current Limit mode and cleared when operating in Constant Voltage mode.
		3	
		4...7	Reserved and Shall be set to zero

6.5.10.1 Output Voltage Field

The **Output Voltage** field **Shall** return the *Source's* output voltage at the time of the request. The output voltage is measured either at the *Source's* receptacle or, if the *Source* has a captive cable, where the voltage is applied to the cable.

The measurement accuracy **Shall** be +/-3% rounded to the nearest 20mV in *SPR PPS Mode*.

If the *Source* does not support the **Output Voltage** field, the field **Shall** be set to 0xFFFF.

6.5.10.2 Output Current Field

The **Output Current** field **Shall** return the *Source's* output current at the time of the request measured at the *Source's* receptacle.

The measurement accuracy **Shall** be +/-150mA.

If the *Source* does not support the **Output Current** field, the field **Shall** be set to 0xFF.

6.5.10.3 Real Time Flags Field

Real Time flags provide a real-time indication of the *Source's* operating state:

- The **PTF** (Present Temperature Flag) **Shall** provide a real-time indication of the *Source's* internal thermal status. If the PTF is not supported, it will be set to zero:
 - Normal indicates that the *Source* is operating within its normal thermal envelope.
 - Warning indicates that the *Source* is over-heating but is not in imminent danger of shutting down.
 - Over Temperature indicates that the *Source* is over heated and will shut down soon or has already shutdown and has sent the **OTP Event** flag in an **Alert Message**.
- The **OMF** (Operating Mode Flag) **Shall** provide a real-time indication of the *SPR PPS Source's* operating mode. When set, the *Source* is operating in **Current Limit** mode; when cleared it is operating **Constant Voltage** mode. This bit **Shall** be set to zero when not in *SPR PPS Mode*.

6.5.11 Country_Codes Message

The **Country_Codes** Message **Shall** be sent in response to a **Get_Country_Codes** Message. The **Country_Codes** Message enables a *Port* to query its *Port Partner* to get a list of alpha-2 country codes as defined in [\[ISO 3166\]](#) for which the *Port Partner* has country specific information.

The **Country_Codes** Message **Shall** contain a 4...26-byte Country Code Data Block (CCDB) whose format **Shall** be as shown in [Figure 6.50, "Country_Codes Message"](#) and [Table 6.63, "Country Codes Data Block \(CCDB\)"](#).

Figure 6.50 Country_Codes Message

Extended Header	CCDB
Data Size = 4-26	(4-26 byte Data Block)

Table 6.63 Country Codes Data Block (CCDB)

Offset	Field	Description		
0	Length	Number of country codes in the <i>Message</i>		
1	Reserved	Shall be set to zero.		
2... Length * 2n	Country Code	Offset	Field	Description
		2	1 st Country Code	First character of the Alpha-2 Country Code defined by [ISO 3166]
		3		Second character of the Alpha-2 Country Code defined by [ISO 3166]
		4	2 nd Country Code	First character of the Alpha-2 Country Code defined by [ISO 3166]
		5		Second character of the Alpha-2 Country Code defined by [ISO 3166]
		...		
		Length * 2n	n th Country Code	

6.5.11.1 Country Code Field

The **Country Code** field **Shall** contain **Length** Country Codes in the Alpha-2 Country Code defined by [\[ISO 3166\]](#).

6.5.12 Country_Info Message

The **Country_Info** Message **Shall** be sent in response to a **Get_Country_Info** Message. The **Country_Info** Message enables a *Port* to get additional country specific information from its *Port Partner*.

The **Country_Info** Message **Shall** contain a 4...26-byte Country Info Data Block (CIDB) whose format **Shall** be as shown in [Figure 6.51, "Country_Info Message"](#) and [Table 6.64, "Country Info Data Block \(CIDB\)"](#).

Figure 6.51 Country_Info Message

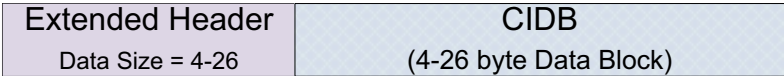


Table 6.64 Country Info Data Block (CIDB)

Offset	Field	Size
0	Country Code	First character of the Alpha-2 Country Code received in the corresponding Get_Country_Info Message.
1		Second character of the Alpha-2 Country Code received in the corresponding Get_Country_Info Message
2...3	Reserved	Shall be set to zero.
4	Country Specific Data	1...22 bytes of content defined by the country's authority.

6.5.12.1 Country Code Field

The **Country Code** field **Shall** contain the Alpha-2 Country Code received in the corresponding **Get_Country_Info** Message.

6.5.12.2 Country Specific Data Field

The **Country Specific Data** field **Shall** contain content defined by and formatted in a manner determined by an official agency of the country indicated in the Country Code field.

If the Country Code field in the **Get_Country_Info** Message is unrecognized then Country Specific Data field **Shall** return the null terminated ASCII text string "Unsupported Code".

6.5.13 Sink_Capabilities_Extended Message

The *Sink_Capabilities_Extended* Message **Shall** be sent in response to a *Get_Sink_Cap_Extended* Message. The *Sink_Capabilities_Extended* Message enables a Sink or a DRP to inform the Source about its Capabilities as a Sink.

The *Sink_Capabilities_Extended* Message **Shall** return a 24-byte Sink Capabilities Extended Data Block (SKEDB) whose format **Shall** be as shown in [Figure 6.52, "Sink_Capabilities_Extended Message"](#) and [Table 6.65, "Sink_Capabilities_Extended Data Block \(SKEDB\)"](#).

Figure 6.52 Sink_Capabilities_Extended Message

Extended Header	SKEDB
Data Size = 24	(24 byte Data Block)

Table 6.65 Sink Capabilities Extended Data Block (SKEDB)

Offset (Byte)	Field	Size (Bytes)	Type	Description	
0	<i>VID</i>	2	Numeric	Vendor ID (assigned by the USB-IF)	
2	<i>PID</i>	2	Numeric	Product ID (assigned by the manufacturer)	
4	<i>XID</i>	4	Numeric	Value provided by the USB-IF assigned to the product	
8	<i>FW Version</i>	1	Numeric	Firmware version number	
9	<i>HW Version</i>	1	Numeric	Hardware version number	
10	<i>SKEDB Version</i>	1	Numeric	SKEDB Version (not the specification Version): <ul style="list-style-type: none">Version 1.0 = 1 Values 0 and 2-255 are Reserved and Shall Not be used.	
11	<i>Load Step</i>	1	Bit Field	Bit	Description
				0...1	<ul style="list-style-type: none">00b: 150mA/μs Load Step (default)01b: 500mA/μs Load Step 11b...10b: Reserved and Shall Not be used.
				2...7	Reserved and Shall be set to zero
12	<i>Sink Load Characteristics</i>	2	Bit Field	Bit	Description
				0...4	Percent overload in 10% increments. Values higher than 25 (11001b) are clipped to 250%. 00000b is the default.
				5...10	Overload period in 20ms when bits 0...4 non-zero.
				1...14	Duty cycle in 5% increments when bits 0...4 are non-zero.
				15	Can tolerate <i>V_{BUS}</i> voltage droop
14	<i>Compliance</i>	1	Bit Field	Bit	Description
				0	Requires <i>LPS Source</i> when set
				1	Requires PS1 <i>Source</i> when set
				2	Requires PS2 <i>Source</i> when set
				3...7	Reserved and Shall be set to zero

Table 6.65 Sink Capabilities Extended Data Block (SKEDB) (Continued)

Offset (Byte)	Field	Size (Bytes)	Type	Description	
15	<i>Touch Temp</i>	1	Value	Temperature conforms to: <ul style="list-style-type: none">0 = Not applicable1 = <i>[IEC 60950-1]</i> (default)2 = <i>[IEC 62368-1]</i> TS13 = <i>[IEC 62368-1]</i> TS2 Note: All other values Reserved	
16	<i>Battery Info</i>	1	Byte	Upper Nibble = Number of Hot Swappable <i>Battery Slots</i> (0...4) Lower Nibble = Number of <i>Fixed Batteries</i> (0...4)	
17	<i>Sink Modes</i>	1	Bit Field	Bit	Description
				0	<i>PPS</i> charging supported
				1	<i>VBUS</i> powered
				2	<i>AC Supply</i> powered
				3	<i>Battery</i> powered
				4	<i>Battery</i> essentially unlimited
				5	<i>AVS</i> Support
6...7	Reserved and Shall be set to zero				
18	<i>SPR Sink Minimum PDP</i>	1	Byte	Bit	Description
				0...6	The <i>PDP</i> of the <i>Source</i> that the <i>Sink</i> requires to operate at its lowest functionality without consuming power from its <i>Battery</i> if it has one.
				7	Reserved and Shall be set to zero
19	<i>SPR Sink Operational PDP</i>	1	Byte	Bit	Description
				0...6	The <i>PDP</i> of the <i>Source</i> that the <i>Sink</i> requires to operate at its normal functionality.
				7	Reserved and Shall be set to zero
20	<i>SPR Sink Maximum PDP</i>	1	Byte	Bit	Description
				0...6	The maximum <i>PDP</i> the <i>Sink</i> will ever request.
				7	Reserved and Shall be set to zero
21	<i>EPR Sink Minimum PDP</i>	1	Byte	The <i>PDP</i> of the <i>Source</i> that the <i>EPR Sink</i> requires to operate at its lowest functionality without consuming power from its <i>Battery</i> if it has one.	
22	<i>EPR Sink Operational PDP</i>	1	Byte	The <i>PDP</i> of the <i>Source</i> that the <i>EPR Sink</i> requires to operate at its normal functionality.	
23	<i>EPR Sink Maximum PDP</i>	1	Byte	The maximum <i>PDP</i> that the <i>EPR Sink</i> will ever request.	

6.5.13.1 Vendor ID (VID) Field

The **VID** field **Shall** contain the 16-bit *Vendor ID (VID)* assigned to the *Sink's* vendor by the USB-IF. If the vendor does not have a *VID*, the **VID** field **Shall** be set to 0xFFFF. *Devices* that have a USB data interface **Shall** report the same *VID* as the *idVendor* in the Standard Device Descriptor (see *[USB 2.0]* and *[USB 3.2]*).

6.5.13.2 Product ID (PID) Field

The **PID** field **Shall** contain the 16-bit Product ID (PID) assigned by the *Sink's* vendor. *Devices* that have a USB data interface **Shall** report the same PID as the *idProduct* in the Standard Device Descriptor (see *[USB 2.0]* and *[USB 3.2]*).

6.5.13.3 XID Field

The **XID** field **shall** contain the 32-bit XID provided by the USB-IF to the vendor who in turns assigns it to a product. If the vendor does not have an XID, then it **shall** return zero in this field (see [\[USB 2.0\]](#) and [\[USB 3.2\]](#)).

6.5.13.4 Firmware Version Field

The **FW Version** field **shall** contain an 8-bit firmware version number assigned to the device by the vendor.

6.5.13.5 Hardware Version Field

The **HW Version** field **shall** contain an 8-bit hardware version number assigned to the device by the vendor.

6.5.13.6 SKEDB Version Field

The **SKEDB Version** field contains the version level of the SKEDB. Currently only Version 1 is defined.

6.5.13.7 Load Step Field

The **Load Step** field contains bits indicating the Load Step Slew Rate and Magnitude that this *Sink* prefers. See [Section 7.1.12.1, "Voltage Regulation Field"](#) for further details.

6.5.13.8 Sink Load Characteristics Field

The *Sink* **shall** report its preferred load characteristics in the **Sink Load Characteristics** field. Regardless of this value, in operation its load **shall Not** exceed the *Capabilities* reported in the **Source_Capabilities_Extended** Message.

6.5.13.9 Compliance Field

The **Compliance** field **shall** contain the types of *Sources* the *Sink* has been tested and certified with (see [Section 7.1.12.3, "Compliance Field"](#)).

6.5.13.10 Touch Temp

The **Touch Temp** field **shall** report the IEC standard used to determine the surface temperature of the *Sink*'s enclosure. Safety limits for the *Sink*'s touch temperature are set in applicable product safety standards (e.g., [\[IEC 60950-1\]](#) or [\[IEC 62368-1\]](#)). The *Sink* **May** report when its touch temperature performance conforms to the TS1 or TS2 limits described in [\[IEC 62368-1\]](#).

6.5.13.11 Battery Info

The **Battery Info** field **shall** report the number of *Fixed Batteries* and *Hot Swappable Battery* slots the *Sink* supports. This field **shall** independently report the number of *Battery Slots* and the number of *Fixed Batteries*. The information reported in the **Battery Info** field **shall** match that reported in the **Number of Batteries/Battery Slots** field of the **Source_Capabilities_Extended** Message.

A *Sink* **shall** have no more than 4 *Fixed Batteries* and no more than 4 *Battery Slots*.

Fixed Batteries **shall** be numbered consecutively from 0 to 3. The number assigned to a given *Fixed Battery* **shall Not** change between *Attach* and *Detach*.

Battery Slots **shall** be numbered consecutively from 4 to 7. The number assigned to a given *Battery Slot* **shall Not** change between *Attach* and *Detach*.

6.5.13.12 Sink Modes

The **Sink Modes** bit field **shall** identify the charging *Capabilities* and the power sources that can be used by the *Sink*. When bit 0 is set, the *Sink* has the ability to use a *PPS Source* for fast charging.

The source of power a *Sink* can use:

- When bit 1 is set, the *Sink* has the ability to be sourced by *VBUS*.
- When bit 2 is set, the *Sink* has the ability to be sourced by an *AC Supply*.
- When bit 3 is set, the *Sink* has the ability to be sourced by a *Battery*.
- When bit 4 is set, the *Sink* has the ability to be sourced by a *Battery* with essentially infinite energy (e.g., a car battery).

Bits 1-4 **may** be set independently of one another. The combination indicates what sources of power the *Sink* can utilize. For example, some *Sinks* are only powered by a *Battery* (e.g., an automobile battery) rather than the more common *AC Supply* and some *Sinks* are only powered from *VBUS* or *VCONN*.

When bit 5 is set, the *Sink* has the ability to support *AVS*.

6.5.13.13 SPR Sink Minimum PDP

The **SPR Sink Minimum PDP** field **shall** contain the minimum power *Source PDP* needed by the *Sink*, rounded up to the next integer, to operate at its lowest level of functionality without requiring power from its *Battery* if present. *Battery* charging may be an opportunistic feature, however this *PDP* **should** be designed for basic functionality, not for charging. The **SPR Sink Minimum PDP** field **shall** be less than or equal to the **SPR Sink Operational PDP**. The value is used by the *Source* to determine whether or not it has sufficient power to minimally support the *Attached Sink*. If the *Sink* is *EPR Capable* and is unable to operate at *PDPs* less than 100W, it **shall** set this field to the minimum power to sustain PD communication.

If the *Sink* is self-powered, such that it doesn't need power from a *Source*, then it **shall** set this field to zero.

The **SPR Sink Minimum PDP** is used to indicate to *Shared Capacity Chargers* the power that **should** be delivered to the *Sink* to guarantee at least basic functionality for the end user.

Possible examples of **SPR Sink Minimum PDP** could be:

- The minimum power a wireless *Charger* would require in order to detect, and deliver the minimum required amount of power to the attached device.
- The power required to have basic functionality by a *Batteryless Sink*,
- On a device with a *Battery*, it can power the minimum functionality of the device

6.5.13.14 SPR Sink Operational PDP

The **SPR Sink Operational PDP** field **shall** contain the *Source PDP* that the manufacturer recommends for the normal functionality of the *Sink*, rounded up to the next integer. This corresponds to the *PDP Rating of Sources* that the *Sink* is designed to operate with (See [Section 10.3.2, "Normative Sink Rules"](#)). The **SPR Sink Operational PDP** field **shall** be sufficient to operate all the *Sink's* functional modes normally AND charge the *Sink's* *Battery* if present. For *Sinks* with a *Battery(s)*, the **SPR Sink Operational PDP** field **shall** correspond to the *PDP Rating* of the *Charger* shipped with the *Sink* or the recommended *Charger's PDP Rating*. If the *Sink* is *EPR Capable* and is unable to operate at *PDPs* less than 100W, it **shall** set the **SPR Sink Minimum PDP** field to the minimum power to sustain PD communication.

If the *Sink* is self-powered, such that it doesn't need power from a *Source*, then it **shall** set this field to zero.

The **SPR Sink Operational PDP** is used to indicate to *Shared Capacity Chargers* that at this power level the user is not expected to receive any performance warning related to the power being supplied to the *Sink*.

6.5.13.15 SPR Sink Maximum PDP

The **SPR Sink Maximum PDP** field **Shall** contain the highest *PDP* the *Sink* will ever request under any operating condition, rounded up to the next integer, including charging its *Battery* if present. The **SPR Sink Maximum PDP** field **Shall Not** be less than the **SPR Sink Operational PDP** field, but **May** be the same. The value is used by the *Source* to determine the maximum amount of power it has to budget for the *Attached Sink*. If the *Sink* is *EPR Capable* and is unable to operate at *PDPs* less than 100W, it **Shall** set this field to the minimum power to sustain PD communication.

If the *Sink* is self-powered, such that it doesn't need power from a *Source*, then it **Shall** set this field to zero.

6.5.13.16 EPR Sink Minimum PDP

The **EPR Sink Minimum PDP** field **Shall** contain the *Source PDP* needed by an *EPR Sink*, rounded up to the next integer, to operate at its lowest level of functionality without requiring power from its *Battery*, if present. *Battery* charging may be an opportunistic feature, however this *PDP* **Should** be designed for basic functionality, not for charging. The **EPR Sink Minimum PDP** field **Shall** be less than or equal to the **EPR Sink Operational PDP** field value. The value is used by the *Source* to determine whether or not it has sufficient power to minimally support the *Attached Sink*. If the *Sink* is not *EPR Capable*, or if the *Sink* is self-powered, such that it doesn't need power from a *Source*, this field **Shall** be set to zero.

The **EPR Sink Minimum PDP** is used to indicate to *Shared Capacity Chargers* the power that **Should** be delivered to the *Sink* to guarantee at least basic functionality for the end user.

Possible examples of **EPR Sink Minimum PDP** could be:

- The power required to have basic functionality by a *Batteryless Sink*,
- On a device with a *Battery*, it can power the minimum functionality of the device.

Note: **EPR Sink Minimum PDP** can be the same as its **SPR Sink Minimum PDP**.

6.5.13.17 EPR Sink Operational PDP

The **EPR Sink Operational PDP** field **Shall** contain the *Source PDP* that the manufacturer recommends for the normal functionality of the *Sink*, rounded up to the next integer. This corresponds to the *PDP Rating* of *EPR Sources* that the *Sink* is designed to operate with (See [Section 10.3.2, "Normative Sink Rules"](#)). The **EPR Sink Operational PDP** **Shall** be sufficient to operate all the *Sink's* functional modes normally AND charge the *Sink's Battery* if present. For *Sinks* with a *Battery(s)*, it **Shall** correspond to the *PDP Rating* of the *Charger* shipped with the *EPR Sink* or the recommended *Charger's PDP Rating*. If the *Sink* is not *EPR Capable*, or if the *Sink* is self-powered, such that it doesn't need power from a *Source*, this field **Shall** be set to zero.

The **EPR Sink Operational PDP** is used to indicate to *Shared Capacity Chargers* that at this power level the user is not expected to receive any performance warning related to the power being supplied to the *Sink*.

6.5.13.18 EPR Sink Maximum PDP

The **EPR Sink Maximum PDP** field **Shall** be highest *PDP* the *EPR Sink* will ever request under any operating condition, rounded up to the next integer, including charging its *Battery* if present. The **EPR Sink Maximum PDP** field **Shall Not** be less than the **EPR Sink Operational PDP**, but **May** be the same. The value is used by the *Source* to determine the maximum amount of power it has to budget for the *Attached Sink*. If the *Sink* is not *EPR Capable*, or if the *Sink* is self-powered, such that it doesn't need power from a *Source*, this field **Shall** be set to zero.

6.5.14 Extended_Control Message

The **Extended_Control** Message extends the *Control Message* space. The **Extended_Control** Message includes one byte of data. The **Extended_Control** Message **Shall** be as shown in [Figure 6.53, "Extended_Control Message"](#) and [Table 6.66, "Extended_Control Data Block \(ECDB\)"](#).

Figure 6.53 Extended_Control Message

Extended Header Data Size = 2	ECDB (2-byte block)
----------------------------------	------------------------

Table 6.66 Extended_Control Data Block (ECDB)

Offset	Field	Value	Description
0	Type	Unsigned Int	<i>Extended_Control Message</i> Type
1	Data	Byte	Shall be set to zero when not used.

The **Extended_Control** Message types are specified in the Type field of the ECDB and are listed in [Table 6.67, "Extended_Control Message Types"](#). The Sent by column indicates entities which **May** send the given Message (Source, Sink or Cable Plug); entities not listed **Shall Not** issue the corresponding Message. The "Valid Start of Packet" column indicates the Messages which **Shall** only be issued in SOP Packets.

Table 6.67 Extended_Control Message Types

Type	Data	Message Type	Sent by	Description	Valid Start of Packet
0		Reserved		All values not explicitly defined are Reserved and Shall Not be used.	
1	Not used	EPR_Get_Source_Cap	Sink or DRP	See Section 6.5.14.1	SOP only
2	Not used	EPR_Get_Sink_Cap	Source or DRP	See Section 6.5.14.2	SOP only
3	Not used	EPR_KeepAlive	Sink	See Section 6.5.14.3	SOP only
4	Not Used	EPR_KeepAlive_Ack	Source	See Section 6.5.14.4	SOP only
5...255		Reserved		All values not explicitly defined are Reserved and Shall Not be used.	

6.5.14.1 EPR_Get_Source_Cap Message

The **EPR_Get_Source_Cap** (EPR Get Source Capabilities) Message **Shall** only be sent by a Port capable of operating as a Sink and that supports EPR Mode to request the Source Capabilities and Dual-Role Power capability of its Port Partner. A Port that can operate as an EPR Source **Shall** respond by returning an **EPR_Source_Capabilities** Message (see [Section 6.5.15.2, "EPR_Source_Capabilities Message"](#)). A Port that does not support EPR Mode as a Source **Shall** return the **Not_Supported** Message.

An EPR Capable Sink Port that is operating in SPR Mode **Shall** treat the **EPR_Source_Capabilities** Message as informational only and **Shall Not** respond with an **EPR_Request** Message.

6.5.14.2 EPR_Get_Sink_Cap Message

The **EPR_Get_Sink_Cap** (EPR Get Sink Capabilities) Message **Shall** only be sent by a Port capable of operating as a Source and that supports EPR Mode to request the Sink Capabilities and Dual-Role Power capability of its Port Partner. A Port that is EPR Capable operating as a Sink **Shall** respond by returning an **EPR_Sink_Capabilities** Message (see [Section 6.5.15.3, "EPR_Sink_Capabilities Message"](#)). A Port that does not support EPR Mode as a Sink **Shall** return the **Not_Supported** Message.

6.5.14.3 EPR_KeepAlive Message

The **EPR_KeepAlive** Message **May** be sent by a Sink operating in *EPR Mode* to meet the requirement for periodic traffic. The Source operating on *EPR Mode* responds by returning an **EPR_KeepAlive_Ack** Message to the Sink. See [Section 6.4.9, "EPR_Request Message"](#) for additional information.

6.5.14.4 EPR_KeepAlive_Ack Message

The **EPR_KeepAlive_Ack** Message **Shall** be sent by a Source operating in *EPR Mode* in response to an **EPR_KeepAlive** Message. See [Section 6.4.9, "EPR_Request Message"](#) for additional information.

6.5.15 EPR Capabilities Message

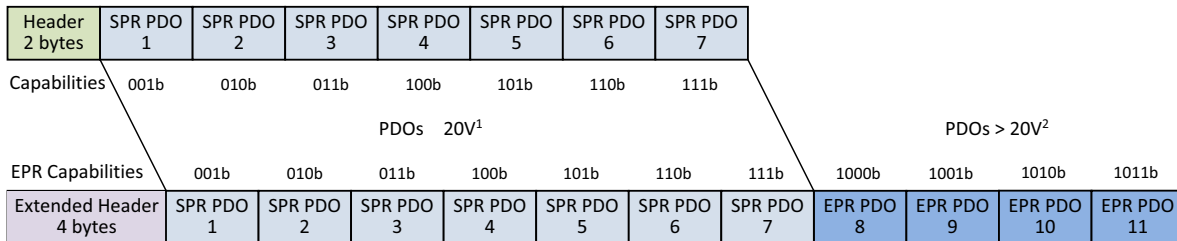
The *EPR Capabilities Message* is an *Extended Capabilities Message* made of *Power Data Objects (PDO)* defined in [Section 6.4.1, "Capabilities Message"](#). It is used to form **EPR_Source_Capabilities** Messages and **EPR_Sink_Capabilities** Messages. Sources expose their EPR power Capabilities by sending an **EPR_Source_Capabilities** Message. Sinks expose their EPR power requirements by returning an **EPR_Sink_Capabilities** Message when requested. Both are composed of a number of 32-bit *Power Data Objects* (see [Table 6.7, "Power Data Object"](#)).

An *EPR Capabilities Message* **Shall** have a 5V Fixed Supply PDO containing the sending Port's information in the first object position followed by up to 10 additional PDOs.

6.5.15.1 EPR Capabilities Message Construction

The *EPR Capabilities Messages* (**EPR_Source_Capabilities** and **EPR_Sink_Capabilities**) are *Extended Messages* with the first seven positions filled with the same *SPR (A)PDOs* returned by the *SPR Capabilities Messages* (**Source_Capabilities** and **Sink_Capabilities**) followed by the *EPR (A)PDOs* starting in the eighth position. See [Figure 6.54, "Mapping SPR Capabilities to EPR Capabilities"](#).

Figure 6.54 Mapping SPR Capabilities to EPR Capabilities



Power Data Objects in the *EPR Capabilities Messages* **Shall** be sent in the following order:

- 1) The *SPR (A)PDOs* as reported in the *SPR Capabilities Message*. The **Number of Data Objects** field in the *Message Header* of the *EPR Capabilities Message* is the same as the **Number of Data Objects** field in the *Message Header* of the *SPR Capabilities Message*.
- 2) If the *SPR Capabilities Message* contains fewer than 7 *PDOs*, the unused *Data Objects* **Shall** be zero filled.
- 3) The *EPR (A)PDOs* as defined in [Section 6.4.1, "Capabilities Message"](#) **Shall** start at *Data Object* position 8 and **Shall** be sent in the following order:
 - a) *Fixed Supply PDOs* that offer 28V, 36V or 48V, if present, **Shall** be sent in voltage order; lowest to highest.
 - b) One *EPR AVS APDO* **Shall** be sent.

6.5.15.2 EPR_Source_Capabilities Message

The **EPR_Source_Capabilities** is an *EPR Capabilities Message* containing a list of *Power Data Objects* that the *EPR Source* is capable of supplying. It is sent by an *EPR Source* in order to convey its *Capabilities* to a *Sink*. An *EPR Source* **Shall** send the **EPR_Source_Capabilities** Message:

- When entering *EPR Mode*
- While in *EPR Modes* when its *Capabilities* change
- In response to an **EPR_Get_Source_Cap** Message
- After a *Soft Reset* while in *EPR Mode*

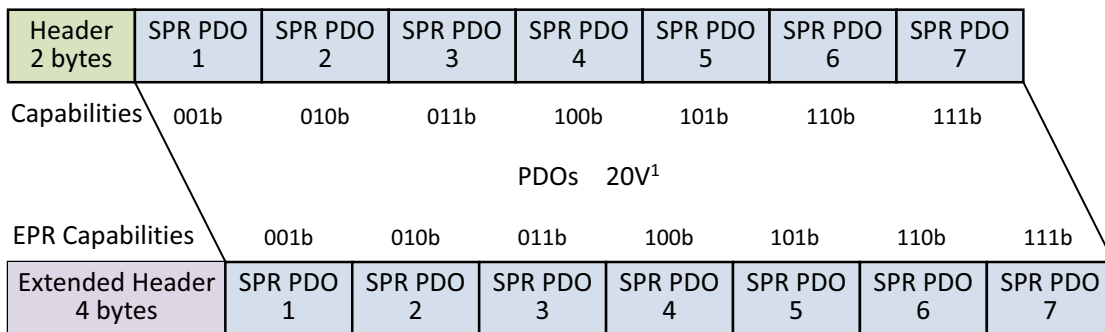
An *EPR Sink* operating in *EPR Mode* **Shall** evaluate every **EPR_Source_Capabilities** Message it receives and **Shall** respond with a **EPR_Request** Message. If its power consumption exceeds the *Source Capabilities*, it **Shall** *Re-negotiate* so as not to exceed the *Source's* most recently *Advertised Source Capabilities*.

While operating in *SPR Mode*, an *EPR Sink* receiving an **EPR_Source_Capabilities** Message in response to an **EPR_Get_Source_Cap** Messages **Shall Not** respond with an **EPR_Request** Message.

The (A)PDOs in an **EPR_Source_Capabilities** Message **Shall** only be requested using the **EPR_Request** Message and only when in *EPR Mode*.

When *Source* wants to exit *EPR Mode*, if not already in power contract with an *SPR (A)PDO*, it **Shall** send an **EPR_Source_Capabilities** Message with no *EPR (A)PDOs* (i.e. seven *SPR (A)PDOs* including any zero padded ones). See [Figure 6.55, "EPR Source Capabilities message with no EPR PDOs"](#).

Figure 6.55 EPR_Source_Capabilities message with no EPR PDOs



¹⁾ See [Section 10 "Power Rules"](#) for rules, on which *SPR (A)PDOs* are allowed to be used for a given *PDP*.

6.5.15.3 EPR_Sink_Capabilities Message

The **EPR_Sink_Capabilities** is an *EPR Capabilities Message* that contains a list of *Power Data Objects* that the *EPR Sink* requires to operate. It is sent by an *EPR Sink* in order to convey its power requirements to an *EPR Source*. The *EPR Sink* **Shall** only send the **EPR_Sink_Capabilities** Message in response to an **EPR_Get_Sink_Cap** Message.

6.5.16 Vendor_Defined_Extended Message

The **Vendor_Defined_Extended Message** (VDEM) is provided to allow vendors to exchange information outside of that defined by this specification using the *Extended Message* format.

A **Vendor_Defined_Extended Message** **Shall** consist of at least one *Vendor Data Object*, the *VDM Header*, and **May** contain up to a maximum of 256 additional data bytes.

To ensure vendor uniqueness of **Vendor_Defined_Extended Messages**, all **Vendor_Defined_Extended Messages** **Shall** contain a **Valid** USB Standard or Vendor ID (SVID) allocated by USB-IF in the *VDM Header*.

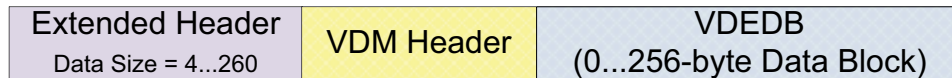
A VDEM does not define any structure and Messages **May** be created in any manner that the vendor chooses.

Vendor_Defined_Extended Messages **Shall Not** be used for direct power *Negotiation*. They **May** however be used to alter *Local Policy*, affecting what is offered or consumed via the normal PD Messages. For example, a **Vendor_Defined_Extended Message** could be used to enable the *Source* to offer additional power via a **Source_Capabilities Message**.

Vendor_Defined_Extended Messages **Shall Not** be used where equivalent functionality is contained in the PD Specification e.g., authentication or firmware update.

The Message format **Shall** be as shown in [Figure 6.56, "Vendor Defined Extended Message"](#).

Figure 6.56 Vendor_Defined_Extended Message



The *VDM Header* **Shall** be the first 4-bytes in a *Vendor Defined Extended Message*. The *VDM Header* provides *Command* space to allow vendors to customize Messages for their own purposes.

The *VDM Header* in the VDEM **Shall** follow the *Unstructured VDM Header* format as defined in [Section 6.4.4.1, "Unstructured VDM"](#).

VDEMs **Shall** only be sent and received after an *Explicit Contract* has been established.

A VDEM AMS **Shall Not** interrupt any other PD AMS.

The VDEM does not define the contents of bits B14...0 in the *VDM Header*. Their definition and use are the sole responsibility of the vendor indicated by the SVID. The *Port Partners* and *Cable Plugs* **Shall** exit any states entered using a VDEM according to the rules defined in [Section 6.4.4.3.4, "Enter Mode Command"](#).

The following rules apply to the use of VDEM Messages:

- VDEMs **Shall Not** be initiated or responded to under any other circumstances than the following:
 - VDEMs **Shall** only be used when an *Explicit Contract* is in place.
 - Prior to establishing an *Explicit Contract* VDEMs **Shall Not** be sent and **Shall** be **Ignored** if received.
 - *Cable Plugs* **Shall Not** initiate VDEMs.

- VDEMs **Shall** only be used during *Modal Operation* in the context of an *Active Mode* i.e., only after the UFP has Ack'ed the **Enter Mode** Command can VDEMs be sent or received. The *Active Mode* and the associated VDEMs **Shall** use the same SVID.
- VDEMs **May** be used with SOP* Packets.
- When a DFP or UFP does not support VDEMs or does not recognize the VID it **Shall** return a **Not_Supported** Message.

Note: Usage of VDEMs with *Chunking* is not recommended since this is less efficient than using *Unstructured VDMs*.

6.6 Timers

All the following timers are defined in terms of bits on the bus regardless of where they are implemented in terms of the logical architecture. This is to ensure a fixed reference for the starting and stopping of timers. It is left to the implementer to ensure that this timing is observed in a real system.

6.6.1 CRCReceiveTimer

The **CRCReceiveTimer** *Shall* be used by the sender's *Protocol Layer* to ensure that a *Message* has not been lost. Failure to receive an acknowledgment of a *Message* (a **GoodCRC** *Message*) whether caused by a bad **GoodCRC** *Message* on the receiving end or by a garbled *Message* within **tReceive** is detected when the **CRCReceiveTimer** expires.

The sender's *Protocol Layer* response when a **CRCReceiveTimer** expires *Shall* be to retry **nRetryCount** times.

Note: *Cable Plugs* do not retry *Messages* and large *Extended Messages* that are not *Chunked* are not retried (see [Section 6.7.2, "Retry Counter"](#)).

Sending of the Preamble corresponding to the retried *Message* *Shall* start within **tRetry** of the **CRCReceiveTimer** expiring.

The **CRCReceiveTimer** *Shall* be started when the last bit of the *Message* **EOP** has been transmitted by the *PHY Layer*. The **CRCReceiveTimer** *Shall* be stopped when the last bit of the **EOP** corresponding to the **GoodCRC** *Message* has been received by the *PHY Layer*.

The *Protocol Layer* receiving a *Message* *Shall* respond with a **GoodCRC** *Message* within **tTransmit** in order to ensure that the sender's **CRCReceiveTimer** does not expire. The **tTransmit** time *Shall* be measured from when the last bit of the *Message* **EOP** has been received by the *PHY Layer* until the first bit of the Preamble of the **GoodCRC** *Message* has been transmitted by the *PHY Layer*.

6.6.2 SenderResponseTimer

The **SenderResponseTimer** *Shall* be used by the sender's *Policy Engine* to ensure that a *Message* requesting a response (e.g., **Get_Source_Cap** *Message*) is responded to within a bounded time of **tSenderResponse**. Failure to receive the expected response is detected when the **SenderResponseTimer** expires.

For *Extended Messages* received as *Chunks*, the **SenderResponseTimer** will also be started and stopped by the *Chunking Rx State Machine*. See [Section 8.3.3.1.1, "SenderResponseTimer State Diagram"](#) for more details of the **SenderResponseTimer** operation.

The *Policy Engine*'s response when the **SenderResponseTimer** expires *Shall* be dependent on the *Message* sent (see [Section 8.3, "Policy Engine"](#)).

The **SenderResponseTimer** *Shall* be started from the time the last bit of the **GoodCRC** *Message* **EOP**, corresponding to the *Message* requesting a response, has been received by the *PHY Layer*.

The **SenderResponseTimer** *Shall* be stopped when the last bit of the **EOP** of the **GoodCRC** *Message*, corresponding to the expected response *Message*, has been transmitted by the *PHY Layer*.

The receiver of a *Message* requiring a response *Shall* respond within **tReceiverResponse** in order to ensure that the sender's **SenderResponseTimer** does not expire.

The **tReceiverResponse** time *Shall* be measured from the time the last bit of the **GoodCRC** *Message* **EOP**, corresponding to the expected request *Message*, has been transmitted by the *PHY Layer* until the first bit of the response *Message* Preamble has been transmitted by the *PHY Layer*.

6.6.3 Capability Timers

Sources and *Sinks* use Capability Timers to determine *Attachment* of a PD Capable device. By periodically sending or requesting *Capabilities*, it is possible to determine PD device *Attachment* when a response is received.

6.6.3.1 SourceCapabilityTimer

Prior to the *First Explicit Contract* a Source **Shall** use the *SourceCapabilityTimer* to periodically send out a *Source_Capabilities* Message every *tTypeCSendSourceCap* while:

- The Port is Attached.
- The Source is not in an active connection with a PD Sink Port.

Whenever there is a *SourceCapabilityTimer* timeout the Source **Shall** send a *Source_Capabilities* Message. It **Shall** then re-initialize and restart the *SourceCapabilityTimer*. The *SourceCapabilityTimer* **Shall** be stopped when the last bit of the *EOP* corresponding to the *GoodCRC* Message has been received by the PHY Layer since a PD connection has been established. At this point, the Source waits for a *Request* Message or a response timeout.

Note: The Source can also stop sending *Source_Capabilities* Message after *nCapsCount* Messages have been sent without a *GoodCRC* Message response (see [Section 6.7.4, "Capabilities Counter"](#)).

See [Section 8.3.3.2, "Policy Engine Source Port State Diagram"](#) for more details of when *Source_Capabilities* Messages are transmitted.

6.6.3.2 SinkWaitCapTimer

The Sink **Shall** support the *SinkWaitCapTimer*.

While in a *Default Contract* or an *Implicit Contract* when a Sink observes an absence of *Source_Capabilities* Messages, after *VBUS* is present, for a duration of *tTypeCSinkWaitCap* the Sink **May** issue *Hard Reset* Signaling in order to restart the sending of *Source_Capabilities* Messages by the Source (see [Section 6.7.4, "Capabilities Counter"](#)) or continue to operate at USB Type-C current.

When a Sink, entering *EPR Mode*, observes an absence of *EPR_Source_Capabilities* Messages, after the *GoodCRC* Message acknowledging the *EPR_Mode* Message with the Action field set to 3 (*Enter Succeeded*), for a duration of *tTypeCSinkWaitCap* the Sink **Shall** issue *Hard Reset* Signaling in order to exit *EPR Mode* (see [Section 6.4.10, "EPR_Mode Message"](#)).

When a Sink, exiting *EPR Mode*, observes an absence of *Source_Capabilities* Messages, after the *GoodCRC* Message acknowledging the *EPR_Mode* Message with the Action field set to 5 (*Exit*), for a duration of *tTypeCSinkWaitCap* the Sink **Shall** issue *Hard Reset* Signaling in order to restart the sending of *Source_Capabilities* Messages by the Source (see [Section 6.7.4, "Capabilities Counter"](#)).

See [Section 8.3.3.3, "Policy Engine Sink Port State Diagram"](#) for more details of when the *SinkWaitCapTimer* is run.

6.6.3.3 tFirstSourceCap

After Port Partners are Attached or after a *Hard Reset* or after a *Power Role Swap* or after a *Fast Role Swap* a Source **Shall** send its first *Source_Capabilities* Message within *tFirstSourceCap* of *VBUS* reaching *vSafe5V*.

After *Soft Reset*, a Source **Shall** send its first *Source_Capabilities* Message within *tFirstSourceCap* after last bit of the *GoodCRC* Message *EOP* corresponding to *Accept* Message.

This ensures that the Sink receives a *Source_Capabilities* Message before the Sink's *SinkWaitCapTimer* expires.

A Source entering *EPR Mode* **Shall** send its first *EPR_Source_Capabilities* Message within *tFirstSourceCap* of the *GoodCRC* Message acknowledging the *EPR_Mode* Message with the Action field set to 3 (*Enter Succeeded*).

A Source exiting *EPR Mode* **Shall** send its first *Source_Capabilities* Message within *tFirstSourceCap* of the *GoodCRC* Message acknowledging the *EPR_Mode* Message with the Action field set to 5 (*Exit*).

6.6.4 Wait Timers and Times

6.6.4.1 SinkRequestTimer

The **SinkRequestTimer** is used to ensure that the time before the next *Sink Request Message*, after a **Wait Message** has been received from the *Source* in response to a *Sink Request Message*, is a minimum of **tSinkRequest** min (see [Section 6.3.12, "Wait Message"](#)).

The **SinkRequestTimer** **Shall** be started when the **EOP** of a **Wait Message** has been received and **Shall** be stopped if any other *Message* is received or during a *Hard Reset*.

The *Sink* **Shall** wait at least **tSinkRequest**, after receiving the **EOP** of a **Wait Message** sent in response to a *Sink Request Message*, before sending a new *Request Message*. Whenever there is a **SinkRequestTimer** timeout the *Sink* **May** send a *Request Message*. It **Shall** then re-initialize and restart the **SinkRequestTimer**.

6.6.4.2 tPRSwapWait

The time before the next **PR_Swap Message**, after a **Wait Message** has been received in response to a **PR_Swap Message** is a minimum of **tPRSwapWait** min (see [Section 6.3.12, "Wait Message"](#)). The *Port* **Shall** wait at least **tPRSwapWait** after receiving the **EOP** of a **Wait Message** sent in response to a **PR_Swap Message**, before sending a new **PR_Swap Message**.

6.6.4.3 tDRSwapWait

The time before the next **DR_Swap Message**, after a **Wait Message** has been received in response to a **DR_Swap Message** is a minimum of **tDRSwapWait** min (see [Section 6.3.12, "Wait Message"](#)). The *Port* **Shall** wait at least **tDRSwapWait** after receiving the **EOP** of a **Wait Message** sent in response to a **DR_Swap Message**, before sending a new **DR_Swap Message**.

6.6.4.4 tVCONNSwapWait

The time before the next **VCONN_Swap Message**, after a **Wait Message** has been received in response to a **VCONN_Swap Message** is a minimum of **tVCONNSwapWait** min (see [Section 6.3.12, "Wait Message"](#)). The *Port* **Shall** wait at least **tVCONNSwapWait** after receiving the **EOP** of a **Wait Message** sent in response to a **VCONN_Swap Message**, before sending a new **VCONN_Swap Message**.

6.6.4.5 tVCONNSwapDelayDFP

The time delay for *DFP* after losing *VCONN Source* role due to an incoming *VCONN Swap* request from *UFP* and before sending the next **VCONN_Swap Message**. The *DFP* **Shall** wait at least **tVCONNSwapDelayDFP** after sending the **EOP** of the **GoodCRC Message** in response to **PS_RDY Message** received at the end of the previous *VCONN Swap AMS*.

6.6.4.6 tVCONNSwapDelayUFP

The time delay for *UFP* after losing *VCONN Source* role due to an incoming *VCONN Swap* request from *DFP* and before sending the next **VCONN_Swap Message**. The *UFP* **Shall** wait at least **tVCONNSwapDelayUFP** after sending the **EOP** of the **GoodCRC Message** in response to **PS_RDY Message** received at the end of the previous *VCONN Swap AMS*.

6.6.4.7 tEnterUSBWait

The time before the next **Enter_USB Message**, after a **Wait Message** has been received in response to a **Enter_USB Message** is a minimum of **tEnterUSBWait** min (see [Section 6.3.12, "Wait Message"](#)). The *DFP* **Shall** wait at least **tEnterUSBWait** after receiving the **EOP** of a **Wait Message** sent in response to an **Enter_USB Message**, before sending a new **Enter_USB Message**.

6.6.5 Power Supply Timers

See [Section 7.3, "Transitions"](#) for diagrams showing the usage of the timers in this section.

6.6.5.1 PSTransitionTimer

The **PSTransitionTimer** is used by the *Policy Engine* to timeout on a **PS_RDY** Message. It is started when a request for new *Source Capabilities* has been accepted and will timeout after **tPSTransition** if a **PS_RDY** Message has not been received. This condition leads to a *Hard Reset* and a return to *USB Default Operation*. The **PSTransitionTimer** relates to the time taken for the *Source* to transition from one voltage, or current level, to another (see [Section 7.1, "Source Requirements"](#)).

The **PSTransitionTimer** **Shall** be started when the last bit of the **GoodCRC** Message **EOP**, corresponding to an **Accept** Message, has been transmitted by the *PHY Layer*. The **PSTransitionTimer** **Shall** be stopped when the last bit of the **GoodCRC** Message **EOP**, corresponding to the **PS_RDY** Message, has been transmitted by the *PHY Layer*.

6.6.5.2 PSSourceOffTimer

6.6.5.2.1 Use during Power Role Swap

The **PSSourceOffTimer** is used by the *Policy Engine* in *Dual-Role Power Device* that is currently acting as a *Sink* to timeout on a **PS_RDY** Message during a *Power Role Swap* AMS. This condition leads to *USB Type-C Error Recovery*.

If a **PR_Swap** Message request has been sent by the *Dual-Role Power Device* currently acting as a *Source* the *Sink* can respond with an **Accept** Message. When the last bit of the **GoodCRC** Message **EOP**, corresponding to this transmitted **Accept** Message, is received by the *Sink's PHY Layer*, then the **PSSourceOffTimer** **Shall** be started.

If a **PR_Swap** Message request has been sent by the *Dual-Role Power Device* currently acting as a *Sink* the *Source* can respond with an **Accept** Message. When the last bit of the **GoodCRC** Message **EOP**, corresponding to this received **Accept** Message, is transmitted by the *Sink's PHY Layer*, then the **PSSourceOffTimer** **Shall** be started.

The **PSSourceOffTimer** **Shall** be stopped when the last bit of the **GoodCRC** Message **EOP**, corresponding to the received **PS_RDY** Message, is transmitted by the *PHY Layer*.

The **PSSourceOffTimer** relates to the time taken for the remote *Dual-Role Power Device* to stop supplying power (see also [Section 7.3.2.1, "Sink Requested Power Role Swap"](#) and [Section 7.3.2.2, "Source Requested Power Role Swap"](#)). The timer **Shall** time out if a **PS_RDY** Message has not been received from the remote *Dual-Role Power Device* within **tPSSourceOff** indicating this has occurred.

6.6.5.2.2 Use during Fast Role Swap

The **PSSourceOffTimer** is used by the *Policy Engine* in *Dual-Role Power Device* that is the *Initial Sink* (currently providing **vSafe5V**) to timeout on a **PS_RDY** Message during a *Fast Role Swap* AMS. This condition leads to *USB Type-C Error Recovery*.

When the **FR_Swap** Message request has been sent by the *Initial Sink*, the *Initial Source* **Shall** respond with an **Accept** Message. When the last bit of the **GoodCRC** Message **EOP**, corresponding to this **Accept** Message is received by the *Initial Sink's PHY Layer*, then the **PSSourceOffTimer** **Shall** be started.

The **PSSourceOffTimer** **Shall** be stopped when

the last bit of the **GoodCRC** Message **EOP**, corresponding to the received **PS_RDY** Message, is transmitted by the *PHY Layer*.

The **PSSourceOffTimer** relates to the time taken for the *Initial Source* to stop supplying power and for *V_{BUS}* to revert to **vSafe5V** (see also [Section 7.2.10, "Fast Role Swap"](#) and [Section 7.3.4, "Transitions Caused by Fast Role Swap"](#)). The timer **Shall** time out if a **PS_RDY** Message has not been received from the *Initial Source* within **tPSSourceOff** indicating this has occurred.

6.6.5.3 PSSourceOnTimer

6.6.5.3.1 Use during Power Role Swap

The **PSSourceOnTimer** is used by the *Policy Engine* in *Dual-Role Power Device* that has just stopped sourcing power and is waiting to start sinking power to timeout on a **PS_RDY** Message during a *Power Role Swap*. This condition leads to *USB Type-C Error Recovery*.

The **PSSourceOnTimer** *Shall* be started when:

- The last bit of the **GoodCRC** Message **EOP**, corresponding to the transmitted **PS_RDY** Message, is received by the *PHY* Layer.
- The **PSSourceOnTimer** *Shall* be stopped when:
- The last bit of the **GoodCRC** Message **EOP**, corresponding to the received **PS_RDY** Message, is transmitted by the *PHY* Layer.

The **PSSourceOnTimer** relates to the time taken for the remote *Dual-Role Power Device* to start sourcing power (see also [Section 7.3.2.1, "Sink Requested Power Role Swap"](#) and [Section 7.3.2.2, "Source Requested Power Role Swap"](#)) and will time out if a **PS_RDY** Message indicating this has not been received within **tPSSourceOn**.

6.6.5.3.2 Use during Fast Role Swap

The **PSSourceOnTimer** is used by the *Policy Engine* in *Dual-Role Power Device* that has just stopped sourcing power and is waiting to start sinking power to timeout on a **PS_RDY** Message during a *Fast Role Swap*. This condition leads to *USB Type-C Error Recovery*.

The **PSSourceOnTimer** *Shall* be started when:

- The last bit of the **GoodCRC** Message **EOP**, corresponding to the transmitted **PS_RDY** Message, is received by the *PHY* Layer.

The **PSSourceOnTimer** *Shall* be stopped when:

- The last bit of the **GoodCRC** Message **EOP**, corresponding to the received **PS_RDY** Message, is transmitted by the *PHY* Layer.

The **PSSourceOnTimer** relates to the time taken for the remote *Dual-Role Power Device* to start sourcing power (see also [Section 7.2.10, "Fast Role Swap"](#) and [Section 7.3.4, "Transitions Caused by Fast Role Swap"](#)) and will time out if a **PS_RDY** Message indicating this has not been received within **tPSSourceOn**.

6.6.6 NoResponseTimer

The **NoResponseTimer** is used by the *Policy Engine* in a *Source* to determine that its *Port Partner* is not responding after a *Hard Reset*. When the **NoResponseTimer** times out, the *Policy Engine* *Shall* issue up to **nHardResetCount** additional *Hard Resets* before determining that the *Port Partner* is non-responsive to *USB Power Delivery* messaging.

If the *Source* fails to receive a **GoodCRC** Message in response to a **Source_Capabilities** Message within **tNoResponse** of:

- The last bit of a **Hard Reset** Signaling being sent by the *PHY* Layer if the **Hard Reset** Signaling was initiated by the *Sink*.
- The last bit of a **Hard Reset** Signaling being received by the *PHY* Layer if the **Hard Reset** Signaling was initiated by the *Source*.

Then the *Source* *Shall* issue additional *Hard Resets* up to **nHardResetCount** times (see [Section 6.8.3, "Hard Reset"](#)).

For a non-responsive device, the *Policy Engine* in a *Source* *May* either decide to continue sending **Source_Capabilities** Messages or to go to non-*USB Power Delivery* operation and cease sending **Source_Capabilities** Messages.

6.6.7 BIST Timers

6.6.7.1 tBISTCarrierMode

tBISTCarrierMode is used to define the maximum time that a *UUT* has to enter *BIST Carrier Mode* when requested by a *Tester*.

A UUT **Shall** enter *BIST Carrier Mode* within **tBISTCarrierMode** of the last bit of the **GoodCRC** Message **EOP**, corresponding to the received the **BIST** Message used to initiate the test, being transmitted by the *PHY Layer*. In *BIST Carrier Mode* when transmitting a continuous carrier signal transmission **Shall** start as soon as the UUT enters *BIST Mode*.

6.6.7.2 BISTContModeTimer

The **BISTContModeTimer** is used by a UUT to ensure that a *Continuous BIST Mode* (i.e., *BIST Carrier Mode*) is exited in a timely fashion. A UUT that has been put into a *Continuous BIST Mode* **Shall** return to normal operation (either **PE_SRC_Transition_to_default**, **PE_SNK_Transition_to_default**, or **PE_CBL_Ready**) within **tBISTContMode** of starting to transmit a continuous carrier signal.

6.6.7.3 tBISTSharedTestMode

tBISTSharedTestMode is used to define the maximum time that a UUT has to enter *BIST Shared Capacity Test Mode* when requested by a *Tester*.

A UUT **Shall** enter *BIST Shared Capacity Test Mode* and send a new **Source_Capabilities** Message from all Ports within the *Shared Capacity Group* within **tBISTSharedTestMode** of the last bit of the **GoodCRC** Message **EOP**, corresponding to the received the **BIST** Message used to initiate the test, being transmitted by the *PHY Layer*.

6.6.8 Power Role Swap Timers

6.6.8.1 SwapSourceStartTimer

The **SwapSourceStartTimer** **Shall** be used by the *New Source*, after a *Power Role Swap* or *Fast Role Swap*, to ensure that it does not send **Source_Capabilities** Message before the *New Sink* is ready to receive the **Source_Capabilities** Message. The *New Source* **Shall Not** send the **Source_Capabilities** Message earlier than **tSwapSourceStart** after the last bit of the **EOP** of **GoodCRC** Message sent in response to the **PS_RDY** Message sent by the *New Source* indicating that its power supply is ready. The *Sink* **Shall** be ready to receive a **Source_Capabilities** Message **tSwapSinkReady** after having sent the last bit of the **EOP** of **GoodCRC** Message sent in response to the **PS_RDY** Message sent by the *New Source* indicating that its power supply is ready.

6.6.9 Soft Reset Timers

6.6.9.1 tSoftReset

A failure to see a **GoodCRC** Message in response to any Message within **tReceive** (after **nRetryCount** retries), when a *Port Pair* is *Connected*, is indicative of a communications failure. This **Shall** cause the *Source* or *Sink* to send a **Soft_Reset** Message, transmission of which **Shall** be completed within **tSoftReset** of the **CRCReceiveTimer** expiring.

6.6.9.2 tProtErrSoftReset

If the *Protocol Error* occurs that causes the *Source* or *Sink* to send a **Soft_Reset** Message, the transmission of the **Soft_Reset** Message **Shall** be completed within **tProtErrSoftReset** of the **EOP** of the **GoodCRC** sent in response to the Message that caused the *Protocol Error*.

6.6.10 Data Reset Timers

6.6.10.1 VCONNDISCHARGEtimer

The **VCONNDISCHARGEtimer** is used by the *Policy Engine* in the *DFP* to ensure the *UFP* actively discharges *VCONN* in a timely manner to ensure the cable will restore *Ra*. Once the *UFP* has discharged *VCONN* below *vRaReconnect* (see [\[USB Type-C 2.4\]](#)) it sends a **PS_RDY** Message (see also [Section 7.1.15, "VCONN Power Cycle"](#)).

If the *DFP* does not receive a **PS_RDY** Message from the *UFP* within **tVCONNSourceDischarge** of the last bit of the **GoodCRC** acknowledging the **Accept** Message in response to the **Data_Reset** Message, the **VCONNDISCHARGEtimer** will time out and the *Policy Engine* **Shall** enter the **ErrorRecovery** State.

6.6.10.2 tDataReset

The *DFP* **shall** complete the *Data_Reset* process (as defined in [Section 6.3.14, "Data_Reset Message"](#)) within *tDataReset* of the last bit of the *GoodCRC* Message *EOP*, corresponding to the *Accept* Message, being transmitted by the *PHY* Layer.

6.6.10.3 DataResetFailTimer

The *DataResetFailTimer* **shall** be used by the *DFP's* Policy Engine to ensure the *Data_Reset* process completes within *tDataResetFail* of the last bit of the *GoodCRC* acknowledging the *Accept* Message in response to the *Data_Reset* Message. If the *DFP's* *DataResetFailTimer* expires, the *DFP* **shall** enter the *ErrorRecovery* State.

6.6.10.4 DataResetFailUFPTimer

The *DataResetFailUFPTimer* **shall** be used by the *UFP's* Policy Engine to ensure the *Data_Reset* process completes within *tDataResetFailUFP* of the last bit of the *GoodCRC* acknowledging the *Accept* Message in response to the *Data_Reset* Message. If the *UFP's* *DataResetFailUFPTimer* expires, the *UFP* **shall** enter the *ErrorRecovery* State.

6.6.11 Hard Reset Timers

6.6.11.1 HardResetCompleteTimer

The *HardResetCompleteTimer* is used by the *Protocol Layer* in the case where it has asked the *PHY* Layer to send *Hard_Reset* Signaling and the *PHY* Layer is unable to send the *Signaling* within a reasonable time due to a non-*Idle* channel. If the *PHY* Layer does not indicate that the *Hard_Reset* Signaling has been sent within *tHardResetComplete* of the *Protocol Layer* requesting transmission, then the *Protocol Layer* **shall** inform the *Policy Engine* that the *Hard_Reset* Signaling has been sent in order to ensure the power supply is reset in a timely fashion.

6.6.11.2 PSHardResetTimer

The *PSHardResetTimer* is used by the *Policy Engine* in a *Source* to ensure that the *Sink* has had sufficient time to process *Hard_Reset* Signaling before turning off its power supply to *VBUS*.

When a *Hard_Reset* occurs the *Source*, stops driving *VCONN*, removes *R_p* from the *CC* pin and starts to transition the *VBUS* voltage to *vSafe0V* either:

- *tPSHardReset* after the last bit of the *Hard_Reset* Signaling has been received from the *Sink* or
- *tPSHardReset* after the last bit of the *Hard_Reset* Signaling has been sent by the *Source*.

See [Section 7.1.5, "Response to Hard Resets"](#).

6.6.11.3 tDRSwapHardReset

If a *DR_Swap* Message is received during *Modal Operation* then a *Hard_Reset* **shall** be initiated by the recipient of the unexpected *DR_Swap* Message; *Hard_Reset* Signaling **shall** be generated within *tDRSwapHardReset* of the *EOP* of the *GoodCRC* sent in response to the *DR_Swap* Message.

6.6.11.4 tProtErrHardReset

If a *Protocol Error* occurs that directly leads to a *Hard_Reset*, the transmission of the *Hard_Reset* Signaling **shall** be completed within *tProtErrHardReset* of the *EOP* of the *GoodCRC* sent in response to the Message that caused the *Protocol Error*.

6.6.12 Structured VDM Timers

6.6.12.1 VDMResponseTimer

The *VDMResponseTimer* **shall** be used by the *Initiator's* Policy Engine to ensure that a *Structured VDM Command* request needing a response (e.g. *Discover Identity* Command request) is responded to within a bounded time of *tVDMSenderResponse*. The *VDMResponseTimer* **shall** be applied to all *Structured VDM Commands* except the

Enter Mode and **Exit Mode** Commands which have their own timers (**VDMModeEntryTimer** and **VDMModeExitTimer** respectively). Failure to receive the expected response is detected when the **VDMResponseTimer** expires.

The Policy Engine's response when the **VDMResponseTimer** expires **Shall** be dependent on the Message sent (see [Section 8.3, "Policy Engine"](#)).

The **VDMResponseTimer** **Shall** be started from the time the last bit of the **GoodCRC** Message **EOP**, corresponding to the VDM Command requesting a response, has been received by the PHY Layer. The **VDMResponseTimer** **Shall** be stopped when the last bit of the **EOP** of the **GoodCRC** Message, corresponding to the expected VDM Command response, has been transmitted by the PHY Layer.

The receiver of a Message requiring a response **Shall** respond within **tVDMReceiverResponse** in order to ensure that the sender's **VDMResponseTimer** does not expire.

The **tVDMReceiverResponse** time **Shall** be measured from the time the last bit of the Message **EOP** has been transmitted by the PHY Layer until the first bit of the response Message Preamble has been transmitted by the PHY Layer.

6.6.12.2 VDMModeEntryTimer

The **VDMModeEntryTimer** **Shall** be used by the Initiator's Policy Engine to ensure that the response to a Structured VDM **Enter Mode** Command request (**ACK** or **NAK** with **ACK** indicating that the requested *Alternate Mode* has been entered) arrives within a bounded time of **tVDMWaitModeEntry**. Failure to receive the expected response is detected when the **VDMModeEntryTimer** expires.

The Policy Engine's response when the **VDMModeEntryTimer** expires is to inform the Device Policy Manager (see [Section 8.3.3.23.1, "DFP Structured VDM Mode Entry State Diagram"](#)).

The **VDMModeEntryTimer** **Shall** be started from the time the last bit of the **EOP** of the **GoodCRC** Message, corresponding to the VDM Command request, has been received by the PHY Layer. The **VDMModeEntryTimer** **Shall** be stopped when the last bit of the **EOP** of the **GoodCRC** Message, corresponding to the expected Structured VDM Command response (**ACK**, **NAK** or **BUSY**), has been transmitted by the PHY Layer.

The receiver of a Message requiring a response **Shall** respond within **tVDMEnterMode** in order to ensure that the sender's **VDMModeEntryTimer** does not expire.

The **tVDMEnterMode** time **Shall** be measured from the time the last bit of the **EOP** of the **GoodCRC** Message, corresponding to VDM Command Request, has been transmitted by the PHY Layer until the first bit of the response Message Preamble has been transmitted by the PHY Layer.

6.6.12.3 VDMModeExitTimer

The **VDMModeExitTimer** **Shall** be used by the Initiator's Policy Engine to ensure that the **ACK** response to a Structured VDM **Exit Mode** Command, indicating that the requested *Alternate Mode* has been exited, arrives within a bounded time of **tVDMWaitModeExit**. Failure to receive the expected response is detected when the **VDMModeExitTimer** expires.

The Policy Engine's response when the **VDMModeExitTimer** expires is to inform the Device Policy Manager (see [Section 8.3.3.23.2, "DFP Structured VDM Mode Exit State Diagram"](#)).

The **VDMModeExitTimer** **Shall** be started from the time the last bit of the **GoodCRC** Message **EOP**, corresponding to the VDM Command requesting a response, has been received by the PHY Layer. The **VDMModeExitTimer** **Shall** be stopped when the last bit of the **GoodCRC** Message **EOP**, corresponding to the expected Structured VDM Command response **ACK**, has been transmitted by the PHY Layer.

The receiver of a Message requiring a response **Shall** respond within **tVDMExitMode** in order to ensure that the sender's **VDMModeExitTimer** does not expire.

The **tVDMExitMode** time **Shall** be measured from the time the last bit of the Message **EOP** has been received by the PHY Layer until the first bit of the response Message Preamble has been transmitted by the PHY Layer.

6.6.12.4 tVDMBusy

The *Initiator* **Shall** wait at least *tVDMBusy*, after receiving a *BUSY* Command response, before repeating the *Structured VDM* request again.

6.6.13 VCONN Timers

6.6.13.1 VCONNOnTimer

The *VCONNOnTimer* is used during a *VCONN Swap*.

The *VCONNOnTimer* **Shall** be started when:

- The last bit of *GoodCRC* Message *EOP*, corresponding to the *Accept* Message, is transmitted or received by the *PHY* Layer.

The *VCONNOnTimer* **Shall** be stopped when:

- The last bit of the *GoodCRC* Message *EOP*, corresponding to the *PS_RDY* Message, is transmitted by the *PHY* Layer.

Prior to sending the *PS_RDY* Message, the Port **Shall** have turned *VCONN* On.

6.6.13.2 tVCONNSourceOff

The *tVCONNSourceOff* time applies during a *VCONN Swap*. The initial *VCONN Source* **Shall** cease sourcing *VCONN* within *tVCONNSourceOff* of the last bit of the *GoodCRC* Message *EOP*, corresponding to the *PS_RDY* Message, being transmitted by the *PHY* Layer.

6.6.14 tCableMessage

Ports compliant with *Revision 3.x* of the specification **Shall Not** wait *tCableMessage* before sending an *SOP' Packet* or *SOP'' Packet* even when communicating using *[USBPD 2.0]* with a *Cable Plug*. This specification defines *Collision Avoidance* mechanisms that obviate the need for this time.

Cable Plugs **Shall** only wait *tCableMessage* before sending an *SOP' Packet* or *SOP'' Packet* when operating at *[USBPD 2.0]*. When operating at Revisions higher than *[USBPD 2.0]* *Cable Plugs* **Shall Not** wait *tCableMessage* before sending an *SOP' Packet* or *SOP'' Packet*.

6.6.15 DiscoverIdentityTimer

The *DiscoverIdentityTimer* is used prior to or during an *Explicit Contract* when discovering whether a *Cable Plug* is PD Capable using *SOP'*. When performing *Cable Discovery* during an *Explicit Contract* the *Discover Identity* Command request **Shall** be sent every *tDiscoverIdentity*. No more than *nDiscoverIdentityCount* *Discover Identity* Messages without a *GoodCRC* Message response **Shall** be sent. If no *GoodCRC* Message response is received after *nDiscoverIdentityCount* *Discover Identity* Command requests have been sent by a Port, the Port **Shall Not** send any further *SOP'/SOP''* Messages.

6.6.16 Collision Avoidance Timers

6.6.16.1 SinkTxTimer

The *SinkTxTimer* is used by the *Protocol Layer* in a *Source* to allow the *Sink* to complete its transmission before initiating an *AMS*.

The *Source* **Shall** wait a minimum of *tSinkTx* after changing R_p from *SinkTxOK* to *SinkTxNG* before initiating an *AMS* by sending a *Message*.

A *Sink* **Shall** only initiate an *AMS* when it has determined that R_p is set to *SinkTxOK*.

6.6.16.2 tSrcHoldsBus

If a transition into the *PE_SRC_Ready* state will result in an immediate transition out of the *PE_SRC_Ready* state within *tSrcHoldsBus* e.g. it is due to a *Protocol Error* that has not resulted in a *Soft Reset*, then the notifications of the end of *AMS* and first *Message* in an *AMS* **May** Not be sent to avoid changing the R_p value unnecessarily.

6.6.17 Fast Role Swap Timers

6.6.17.1 tFRSwap5V

The *tFRSwap5V* time **Shall** be measured from:

- The later of:
 - The last bit of the *GoodCRC* Message *EOP*, corresponding to the *Accept* Message or
 - *VBUS* being within *vSafe5V*.
- Until the first bit of the response *PS_RDY* Message Preamble has been transmitted by the *PHY Layer*.

During a *Fast Role Swap*, the *Initial Source* **Shall** start the *PS_RDY* Message within *tFRSwap5V* after both:

- The *Initial Source* has sent the *Accept* Message, and
- *VBUS* is at or below *vSafe5V*.

6.6.17.2 tFRSwapComplete

During a fast-role swap, the *Initial Sink* **Shall** respond with a the *PS_RDY* Message within *tFRSwapComplete* after it has received the *PS_RDY* Message from the *Initial Source*. The *tFRSwapComplete* time **Shall** be measured from the time the last bit of the *GoodCRC* Message *EOP*, corresponding to the *PS_RDY* Message, has been transmitted by the *PHY Layer* until the first bit of the response *PS_RDY* Message Preamble has been transmitted by the *PHY Layer*.

6.6.17.3 tFRSwapInit

That last bit of the *EOP* of the *FR_Swap* Message **Shall** be transmitted by the *New Source* no later than *tFRSwapInit* after the *Fast Role Swap Request* has been detected (see [Section 5.8.6.3, "Fast Role Swap Detection"](#)).

6.6.18 Chunking Timers

6.6.18.1 ChunkingNotSupportedTimer

The *ChunkingNotSupportedTimer* is used by a *Source* or *Sink* which does not support multi-chunk *Chunking* but has received a *Message Chunk*.

The *ChunkingNotSupportedTimer* **Shall** be started when:

- The last bit of the *GoodCRC* Message *EOP*, corresponding to a *Message Chunk* of a multi-chunk *Message*, is transmitted by the *PHY Layer*. The *Policy Engine* **Shall Not** send its *Not_Supported* Message before the *ChunkingNotSupportedTimer* expires.

6.6.18.2 ChunkSenderRequestTimer

The *ChunkSenderRequestTimer* is used during a *Chunked Message* transmission.

The *ChunkSenderRequestTimer* **Shall** be used by the sender's *Chunking* state machine to ensure that a *Chunk Response* is responded to within a bounded time of *tChunkSenderRequest*. Failure to receive the expected response is detected when the *ChunkSenderRequestTimer* expires.

The *ChunkSenderRequestTimer* **Shall** be started when:

- The last bit of the *GoodCRC* Message *EOP*, corresponding to the *Chunk Response Message*, is received by the *PHY Layer*.

The **ChunkSenderRequestTimer** *Shall* be stopped when:

- The last bit of the **EOP** of the **GoodCRC** Message, corresponding to the *Chunk Request Message*, is transmitted by the *PHY Layer*.
- A Message other than a *Chunk Request* is received from the *Protocol Layer Rx*.

The receiver of a *Chunk Response* requiring a *Chunk Request* *Shall* respond with a *Chunk Request* within **tChunkReceiverRequest** in order to ensure that the sender's **ChunkSenderRequestTimer** does not expire.

The **tChunkReceiverRequest** time *Shall* be measured from the time the last bit of the **EOP** of the **GoodCRC** Message, corresponding to the *Chunk Response Message*, has been transmitted by the *PHY Layer* until the first bit of the response Message Preamble has been transmitted by the *PHY Layer*.

6.6.18.3 ChunkSenderResponseTimer

The **ChunkSenderResponseTimer** is used during a *Chunked Message* transmission.

The **ChunkSenderResponseTimer** *Shall* be used by the sender's *Chunking* state machine to ensure that a *Chunk Request* is responded to within a bounded time of **tChunkSenderResponse**. Failure to receive the expected response is detected when the **ChunkSenderResponseTimer** expires.

The **ChunkSenderResponseTimer** *Shall* be started when:

- The last bit of the **GoodCRC** Message **EOP**, corresponding to the *Chunk Request Message*, is received by the *PHY Layer*.

The **ChunkSenderResponseTimer** *Shall* be stopped when:

- The last bit of the **GoodCRC** Message **EOP**, corresponding to the *Chunk Response Message*, is transmitted by the *PHY Layer*.
- A Message other than a *Chunk* is received from the *Protocol Layer*.

The receiver of a *Chunk Request* requiring a *Chunk Response* *Shall* respond with a *Chunk Response* within **tChunkReceiverResponse** in order to ensure that the sender's **ChunkSenderResponseTimer** does not expire.

The **tChunkReceiverResponse** time *Shall* be measured from the time the last bit of the **EOP** of the **GoodCRC** Message, corresponding to the *Chunk Request Message*, has been transmitted by the *PHY Layer* until the first bit of the response Message Preamble has been transmitted by the *PHY Layer*.

6.6.19 Programmable Power Supply Timers

6.6.19.1 SinkPPSPeriodicTimer

The **SinkPPSPeriodicTimer** *Shall* be used by the Sink's Policy Engine to ensure that communication between the Sink and Source occurs within a bounded time of **tPPSRequest** when in *SPR PPS Mode*. In the absence of any other traffic, a **Request** Message requesting an *SPR PPS APDO* is sent periodically as a keep alive mechanism.

SinkPPSPeriodicTimer *Shall* be re-initialized and restarted on transmission, by the *PHY Layer*, of the last bit of the **GoodCRC** Message **EOP**, corresponding to any received Message, that causes the Sink to enter the **PE_SNK_Ready** state.

The Sink *Shall* stop the **SinkPPSPeriodicTimer** on transmission, by the *PHY Layer*, of the last bit of the **GoodCRC** Message **EOP**, corresponding to any Message, or the last bit of any Signaling is received, by the *PHY Layer*, from the Source and by the Sink that causes the Sink to leave the **PE_SNK_Ready** state.

6.6.19.2 SourcePPSCommTimer

The **SourcePPSCommTimer** *Shall* be used by the Source's Policy Engine to ensure that communication between the Sink and Source occurs within a bounded time of **tPPSTimeout** when in *SPR PPS Mode*. In the absence of any other traffic, a **Request** Message requesting an *SPR PPS APDO* is received periodically as a keep alive mechanism.

SourcePPSCmmTimer *Shall* be re-initialized and restarted when, after receiving any *Message* that causes the *Source* to enter the **PE_SRC_Ready** state, the last bit of the corresponding **GoodCRC** *Message EOP* is transmitted by the *PHY Layer*.

The *Source* *Shall* stop the **SourcePPSCmmTimer** when:

- After receiving any *Message* that causes the *Source* to leave the **PE_SRC_Ready** state, the last bit of the corresponding **GoodCRC** *Message EOP* is sent by the *PHY Layer*, or
- The last bit of any *Signaling* is received by the *PHY Layer* from the *Sink* by the *Source* that causes the *Source* to leave the **PE_SRC_Ready** state.

When the **SourcePPSCmmTimer** times out the *Source* *Shall* issue **Hard Reset** *Signaling*.

6.6.20 tEnterUSB

The *DFP* *Shall* send the **Enter_USB** *Message* within **tEnterUSB** of either:

- The last bit of the **GoodCRC** acknowledging the **Data_Reset_Complete** *Message* in response to the **Data_Reset** *Message* or
- A PD Connection, specifically the last bit of the **GoodCRC** acknowledging the **Source_Capabilities** *Message* after the initial entry into the **PE_SRC_Send_Capabilities** state or
- The last bit of the **GoodCRC** acknowledging the **Accept** *Message* in response to the **DR_Swap** *Message*

Failure by the *DFP* to meet this timeout parameter can result in the ports not transitioning into **[USB4]** operation. Any *AMS* initiated by the *UFP* prior to receiving the **Enter_USB** *Message* will delay reception of the **Enter_USB** *Message* and **[USB4]** operation, therefore a USB4[®]-capable *UFP* *Should Not* initiate any *AMS* until the *DFP* has been given time to send the **Enter_USB** *Message*.

6.6.21 EPR Timers

6.6.21.1 SinkEPREnterTimer Timer

The **SinkEPREnterTimer** is used to ensure the *EPR Mode* entry process completes within **tEnterEPR**. The *Sink* *Shall* start the timer when it sees the last bit of the **GoodCRC** *Message* in response to the **EPR_Mode** *Message* with the Action field set to 1 (**Enter**). The *Sink* *Shall* stop the timer when the last bit of the corresponding **GoodCRC** *Message EOP*, corresponding to the received **EPR_Mode** *Message* with the Action field set to 3 (**Enter Succeeded**), has been transmitted by the *PHY Layer*. If the timer expires the *Sink* *Shall* send a **Soft_Reset** *Message*.

6.6.21.2 SinkEPRKeepAlive Timer

The **SinkEPRKeepAliveTimer** *Shall* be used by the *Sink's Policy Engine* to ensure that communication between the *Sink* and *Source* occurs within a bounded time of **tSinkEPRKeepAlive**. The *Sink* *Shall* initialize and run this timer upon entry into the **PE_SNK_Ready** State when in *EPR Mode* and *Shall* stop it upon exit from the **PE_SNK_Ready** when in *EPR Mode*.

While operating in *EPR Mode*, the *Sink* *Shall* stop the **SinkEPRKeepAliveTimer** timer whenever:

- The last bit of the **GoodCRC** *Message EOP*, in response any *Message* from the *Source*, is transmitted by the *PHY Layer*.
- The *PHY Layer* receives the last bit of the **GoodCRC** *Message EOP* in response to any *Message* sent to the *Source*.

If the timer expires the *Sink* *Shall* send an **EPR_KeepAlive** *Message*.

6.6.21.3 SourceEPRKeepAlive Timer

The **SourceEPRKeepAliveTimer** *Shall* be used by the *Source's Policy Engine* to ensure that communication between the *Sink* and *Source* occurs within a bounded time of **tSourceEPRKeepAlive**. The *Source* *Shall* initialize

and run this timer upon entry into the **PE_SRC_Ready** State when in *EPR Mode* and **Shall** disable it upon exit from the **PE_SRC_Ready** State when *EPR Mode*.

While operating in *EPR Mode*, the Source **Shall** stop the **SourceEPRKeepAliveTimer** timer whenever:

- The last bit of the **GoodCRC** Message **EOP**, in response any *Message* from the *Sink*, is transmitted by the *PHY Layer*.
- The *PHY Layer* receives the last bit of the **GoodCRC** Message **EOP** in response to any *Message* sent to the Source.

If the timer expires the Source **Shall** send **Hard Reset** Signaling.

6.6.21.4 **tEPRSourceCableDiscovery**

After *Port Partners* are *Attached* or after a *Hard Reset* or after a *Power Role Swap* or after a *Fast Role Swap* an *EPR Source* **Shall** discover the *Cable Plug* within **tEPRSourceCableDiscovery** of entering the *First Explicit Contract*.

The *EPR Source* **Shall** send the **Discover Identity REQ** Command, to the *Cable Plug*, within **tEPRSourceCableDiscovery** of receiving the **GoodCRC** Message acknowledging the **PS_RDY** Message as part of the *Explicit Contract Negotiation*.

Note: If the *EPR Source* is not the *VCONN Source*, **tEPRSourceCableDiscovery**, will also include the time needed for the *VCONN Swap*.

6.6.22 Time Values and Timers

[Table 6.68, "Time Values"](#) summarizes the values for the timers listed in this section. For each Timer Value, a given implementation **shall** pick a fixed value within the range specified. [Table 6.69, "Timers"](#) lists the timers.

Table 6.68 Time Values

Parameter	Value (min)	Value (Nom)	Value (max)	Units	Reference
<i>tACTempUpdate</i>			500	ms	Section 6.5.2.2.1
<i>tBISTContMode</i>	30	45	60	ms	Section 6.6.7.2
<i>tBISTCarrierMode</i>			300	ms	Section 6.6.7.1
<i>tBISTSharedTestMode</i>			1	s	Section 6.6.7.3
<i>tCableMessage</i>	750			µs	Section 6.6.14
<i>tCapabilitiesMismatchResponse</i>			2	s	Section 6.4.2.3
<i>tChunkingNotSupported</i>	40	45	50	ms	Section 6.6.18.1
<i>tChunkReceiverRequest</i>			15	ms	Section 6.6.18.2
<i>tChunkReceiverResponse</i>			15	ms	Section 6.6.18.3
<i>tChunkSenderRequest</i>	24	27	30	ms	Section 6.6.18.2
<i>tChunkSenderResponse</i>	24	27	30	ms	Section 6.6.18.3
<i>tDataReset</i>	200	225	250	ms	Section 6.6.10.2
<i>tDataResetFail</i>	300		400	ms	Section 6.6.10.3
<i>tDataResetFailUFP</i>	450		550	ms	Section 6.6.10.4
<i>tDiscoverIdentity</i>	40		50	ms	Section 6.6.14
<i>tDRSwapHardReset</i>			15	ms	Section 6.6.11.3
<i>tDRSwapWait</i>	100			ms	Section 6.6.4.3
<i>tEnterUSB</i>			500	ms	Section 6.6.20
<i>tEnterUSBWait</i>	100			ms	Section 6.6.4.7
<i>tEnterEPR</i>	450	500	550	ms	Section 6.6.21.1
<i>tEPRSourceCableDiscovery</i>			2	s	Section 6.6.21.4
<i>tFirstSourceCap</i>			250	ms	Section 6.6.3.3
<i>tFRSwap5V</i>			15	ms	Section 6.6.17.1
<i>tFRSwapComplete</i>			15	ms	Section 6.6.17.2
<i>tFRSwapInit</i>			15	ms	Section 6.6.17.3
<i>tHardReset</i>			5	ms	Section 6.3.13
<i>tHardResetComplete</i>	4	4.5	5	ms	Section 6.6.9
<i>tSourceEPRKeepAlive</i>	0.750	0.875	1.000	s	Section 6.6.21.3
<i>tSinkEPRKeepAlive</i>	0.250	0.375	0.500	s	Section 6.6.21.2
<i>tNoResponse</i>	4.5	5.0	5.5	s	Section 6.6.6
<i>tPPSRequest</i>			10	s	Section 6.6.19.1
<i>tPPSTimeout</i>	12.0	13.5	15.0	s	Section 6.6.19.2
<i>tProtErrHardReset</i>			15	ms	Section 6.6.11.4
<i>tProtErrSoftReset</i>			15	ms	Section 6.6.9.2
<i>tPRSwapWait</i>	100			ms	Section 6.6.4.2
<i>tPSHardReset</i>	25	30	35	ms	Section 6.6.11.2

Table 6.68 Time Values (Continued)

Parameter		Value (min)	Value (Nom)	Value (max)	Units	Reference
<i>tPSSourceOff</i>	<i>SPR Mode</i>	750	835	920	ms	Section 6.6.5.2
	<i>EPR Mode</i>	1120	1260	1400		
<i>tPSSourceOn</i>	<i>SPR Mode</i>	390	435	480	ms	Section 6.6.5.3
<i>tPSTransition</i>	<i>SPR Mode</i>	450	500	550	ms	Section 6.6.5.1
	<i>EPR Mode</i>	830	925	1020		
<i>tReceive</i>		0.9	1.0	1.1	ms	Section 6.6.1
<i>tReceiverResponse</i>				15	ms	Section 6.6.2
<i>tRetry</i>				195	µs	Section 6.6.1
<i>tSenderResponse</i>		27	30	33	ms	Section 6.6.2
<i>tSinkDelay</i>				5	ms	Section 5.7
<i>tSinkRequest</i>		100			ms	Section 6.6.4.1
<i>tSinkTx</i>		16	18	20	ms	Section 6.6.16
<i>tSoftReset</i>				15	ms	Section 6.8.1
<i>tSrcHoldsBus</i>				50	ms	Section 8.3.3.2
<i>tSwapSinkReady</i>				15	ms	Section 6.6.8.1
<i>tSwapSourceStart</i>		20			ms	Section 6.6.8.1
<i>tTransmit</i>				195	µs	Section 6.6.1
<i>tTypeCSendSourceCap</i>		100	150	200	ms	Section 6.6.3.1
<i>tTypeCSinkWaitCap</i>		310	465	620	ms	Section 6.6.3.2
<i>tVCONNSourceDischarge</i>		160	200	240	ms	Section 6.6.10.1
<i>tVCONNSourceOff</i>				25	ms	Section 6.6.13
<i>tVCONNSourceOn</i>				50	ms	Section 6.3.11
<i>tVCONNSourceTimeout</i>		100	150	200	ms	Section 6.6.13
<i>tVCONNSwapWait</i>		100			ms	Section 6.6.4.4
<i>tVCONNSwapDelayDFP</i>		100			ms	Section 6.6.4.5
<i>tVCONNSwapDelayUFP</i>		500			ms	Section 6.6.4.6
<i>tVDMBusy</i>		50			ms	Section 6.6.12.4
<i>tVDMEnterMode</i>				25	ms	Section 6.6.12.2
<i>tVDMLExitMode</i>				25	ms	Section 6.6.12.3
<i>tVDMReceiverResponse</i>				15	ms	Section 6.6.12.1
<i>tVDMSenderResponse</i>		24	27	30	ms	Section 6.6.12.1
<i>tVDMWaitModeEntry</i>		40	45	50	ms	Section 6.6.12.2
<i>tVDMWaitModeExit</i>		40	45	50	ms	Section 6.6.12.3

Table 6.69 Timers

Timer	Parameter	Used By	Reference
<i>BISTContModeTimer</i>	<i>tBISTContMode</i>	Policy Engine	Section 6.6.7.2
<i>ChunkingNotSupportedTimer</i>	<i>tChunkingNotSupported</i>	Policy Engine	Section 6.6.18.1
<i>ChunkSenderRequestTimer</i>	<i>tChunkSenderRequest</i>	Protocol Layer	Section 6.6.18.2
<i>ChunkSenderResponseTimer</i>	<i>tChunkSenderResponse</i>	Protocol Layer	Section 6.6.18.3
<i>CRCReceiveTimer</i>	<i>tReceive</i>	Protocol Layer	Section 6.6.1
<i>DataResetFailTimer</i>	<i>tDataResetFail</i>	Policy Engine	Section 6.6.10.3
<i>DataResetFailUFPTimer</i>	<i>tDataResetFailUFP</i>	Policy Engine	Section 6.6.10.4
<i>DiscoverIdentityTimer</i>	<i>tDiscoverIdentity</i>	Policy Engine	Section 6.6.15
<i>HardResetCompleteTimer</i>	<i>tHardResetComplete</i>	Protocol Layer	Section 6.6.9
<i>NoResponseTimer</i>	<i>tNoResponse</i>	Policy Engine	Section 6.6.6
<i>PSHardResetTimer</i>	<i>tPSHardReset</i>	Policy Engine	Section 6.6.11.2
<i>PSSourceOffTimer</i>	<i>tPSSourceOff</i>	Policy Engine	Section 6.6.5.2
<i>PSSourceOnTimer</i>	<i>tPSSourceOn</i>	Policy Engine	Section 6.6.5.3
<i>PSTransitionTimer</i>	<i>tPSTransition</i>	Policy Engine	Section 6.6.5.1
<i>SenderResponseTimer</i>	<i>tSenderResponse</i>	Policy Engine	Section 6.6.2
<i>SinkEPREnterTimer</i>	<i>tEnterEPR</i>	Policy Engine	Section 6.6.21.1
<i>SinkEPRKeepAliveTimer</i>	<i>tSinkEPRKeepAlive</i>	Policy Engine	Section 6.6.21.2
<i>SinkPPSPeriodicTimer</i>	<i>tPPSRequest</i>	Policy Engine	Section 6.6.19.1
<i>SinkRequestTimer</i>	<i>tSinkRequest</i>	Policy Engine	Section 6.6.4
<i>SinkWaitCapTimer</i>	<i>tTypeCSinkWaitCap</i>	Policy Engine	Section 6.6.3.2
<i>SourceCapabilityTimer</i>	<i>tTypeCSendSourceCap</i>	Policy Engine	Section 6.6.3.1
<i>SourceEPRKeepAliveTimer</i>	<i>tSourceEPRKeepAlive</i>	Policy Engine	Section 6.6.21.3
<i>SourcePPSCommTimer</i>	<i>tPPSTimeout</i>	Policy Engine	Section 6.6.19.2
<i>SinkTxTimer</i>	<i>tSinkTx</i>	Protocol Layer	Section 6.6.16
<i>SwapSourceStartTimer</i>	<i>tSwapSourceStart</i>	Policy Engine	Section 6.6.8.1
<i>VCONNDischargeTimer</i>	<i>tVCONNSourceDischarge</i>	Policy Engine	Section 6.6.10.1
<i>VCONNOnTimer</i>	<i>tVCONNSourceTimeout</i>	Policy Engine	Section 6.6.13.1
<i>VDMMModeEntryTimer</i>	<i>tVDMWaitModeEntry</i>	Policy Engine	Section 6.6.12.2
<i>VDMMModeExitTimer</i>	<i>tVDMWaitModeExit</i>	Policy Engine	Section 6.6.12.3
<i>VDMResponseTimer</i>	<i>tVDMSenderResponse</i>	Policy Engine	Section 6.6.12.1

6.7 Counters

6.7.1 MessageID Counter

The **MessageIDCounter** is a rolling counter, ranging from 0 to **nMessageIDCount**, used to detect duplicate Messages. This value is used for the **MessageID** field in the *Message Header* of each transmitted *Message*.

Each *Port* **Shall** maintain a copy of the last **MessageID** value received from its *Port Partner*. Devices that support multiple ports, such as *Hubs*, **Shall** maintain copies of the last **MessageID** on a per *Port* basis. A *Port* which communicates using *SOP* Packets* **Shall** maintain copies of the last **MessageID** for each type of *SOP** it uses.

The transmitter **Shall** use the **MessageID** in a **GoodCRC** *Message* to verify that a particular *Message* was received correctly. The receiver **Shall** use the **MessageID** to detect duplicate Messages.

6.7.1.1 Transmitter Usage

The Transmitter **Shall** use the **MessageID** as follows:

- Upon receiving either **Hard Reset** Signaling, or a **Soft_Reset** *Message*, the transmitter **Shall** set its **MessageIDCounter** to zero and re-initialize its retry mechanism.
- If a **GoodCRC** *Message* with a **MessageID** matching the **MessageIDCounter** is not received before the **CRCReceiveTimer** expires, it **Shall** retry the same *Packet* up to **nRetryCount** times using the same **MessageID**.
- If a **GoodCRC** *Message* is received with a **MessageID** matching the current **MessageIDCounter** before the **CRCReceiveTimer** expires, the transmitter **Shall** re-initialize its retry mechanism and increment its **MessageIDCounter**.
- If the *Message* is aborted by the *Policy Engine*, the transmitter **Shall** delete the *Message* from its transmit buffer, re-initialize its retry mechanism and increment its **MessageIDCounter**.

6.7.1.2 Receiver Usage

The Receiver **Shall** use the **MessageID** as follows:

- When the first good *Packet* is received after a reset, the receiver **Shall** store a copy of the received **MessageID** value.
- For subsequent Messages, if **MessageID** value in a received *Message* is the same as the stored value, the receiver **Shall** return a **GoodCRC** *Message* with that **MessageID** value and drop the *Message* (this is a retry of an already received *Message*).

Note: This **Shall Not** apply to the **Soft_Reset** *Message* which always has a **MessageID** value of zero.

- If **MessageID** value in the received *Message* is different than the stored value, the receiver **Shall** return a **GoodCRC** *Message* with the new **MessageID** value, store a copy of the new **MessageID** value and process the *Message*.

6.7.2 Retry Counter

The **RetryCounter** is used by a *Port* whenever there is a *Message* transmission failure (timeout of **CRCReceiveTimer**). If the **nRetryCount** retry fails, then the link **Shall** be reset using the *Soft Reset* mechanism.

The following rules apply to retries when there is a *Message* transmission failure (see also [Section 6.12.2.2, "Protocol Layer Message Transmission"](#)):

- *Cable Plugs* **Shall Not** retry Messages.
- *Extended Messages* of **Data Size** > **MaxExtendedMsgLegacyLen** that are not *Chunked* (**Chunked** flag set to zero) **Shall Not** be retried.

- Extended Messages of **Data Size** \leq **MaxExtendedMsgLegacyLen** (**Chunked** flag set to zero or one) **Shall** be retried.
- Extended Messages of **Data Size** $>$ **MaxExtendedMsgLegacyLen** that are **Chunked** (**Chunked** flag set to one) individual Chunks **Shall** be retried.

When Messages are not retried, then the **RetryCounter** is not used. Higher layer protocols are expected to accommodate Message delivery failure or failure to receive a **GoodCRC** Message.

6.7.3 Hard Reset Counter

The **HardResetCounter** is used to retry the **Hard Reset** whenever there is no response from the remote device (see [Section 6.6.6, "NoResponseTimer"](#)). Once the **Hard Reset** has been retried **nHardResetCount** times then it **Shall** be assumed that the remote device is non-responsive.

6.7.4 Capabilities Counter

The **CapsCounter** is used to count the number of **Source_Capabilities** Messages which have been sent by a **Source** at power up or after a **Hard Reset**. Implementation of the **CapsCounter** is **Optional** but **May** be used by any **Source** which wishes to preserve power by not sending **Source_Capabilities** Messages after a period of time.

When the **CapsCounter** is implemented and the **Source** detects that a **Sink** is **Attached** then after **nCapsCount** **Source_Capabilities** Messages have been sent the **Source** **Shall** decide that the **Sink** is non-responsive, stop sending **Source_Capabilities** Messages and disable PD.

A **Sink** **Shall** use the **SinkWaitCapTimer** to trigger the resending of **Source_Capabilities** Messages by a USB Power Delivery capable **Source** which has previously stopped sending **Source_Capabilities** Messages. Any **Sink** which is **Attached** and does not detect a **Source_Capabilities** Message, **Shall** issue **Hard Reset** Signaling when the **SinkWaitCapTimer** times out in order to reset the **Source**. Resetting the **Source** **Shall** also reset the **CapsCounter** and restart the sending of **Source_Capabilities** Messages.

6.7.5 Discover Identity Counter

When sending **Discover Identity** Messages to a **Cable Plug** a **Port** **Shall** maintain a count of Messages sent (**DiscoverIdentityCounter**). No more than **nDiscoverIdentityCount** **Discover Identity** Messages **Shall** be sent by the **Port** without receiving a **GoodCRC** Message response. A **VCONN Swap** **Shall** reset the **DiscoverIdentityCounter**.

6.7.6 VDMBusyCounter

When sending **Responder BUSY** responses to a Structured **Vendor_Defined** Message a **UFP** or **Cable Plug** **Shall** maintain a count of Messages sent (**VDMBusyCounter**). No more than **nBusyCount** **Responder BUSY** responses **Shall** be sent. The **VDMBusyCounter** **Shall** be reset on sending a non-**BUSY** response. Products wishing to meet [\[USB Type-C 2.4\]](#) requirements for **Alternate Mode** entry **Should** use an **nBusyCount** of 1.

6.7.7 Counter Values and Counters

[Table 6.70, "Counter Parameters"](#) lists the counters used in this section and [Table 6.71, "Counters"](#) shows the corresponding parameters.

Table 6.70 Counter Parameters

Parameter	Value	Reference
<i>nBusyCount</i>	5	Section 6.7.6
<i>nCapsCount</i>	50	Section 6.7.4
<i>nDiscoverIdentityCount</i>	20	Section 6.7.5
<i>nHardResetCount</i>	2	Section 6.7.3
<i>nMessageIDCount</i>	7	Section 6.7.1
<i>nRetryCount</i>	2	Section 6.7.2

Table 6.71 Counters

Counter	Max	Reference
<i>CapsCounter</i>	<i>nCapsCount</i>	Section 6.7.4
<i>DiscoverIdentityCounter</i>	<i>nDiscoverIdentityCount</i>	Section 6.7.5
<i>HardResetCounter</i>	<i>nHardResetCount</i>	Section 6.7.3
<i>MessageIDCounter</i>	<i>nMessageIDCount</i>	Section 6.7.1
<i>RetryCounter</i>	<i>nRetryCount</i>	Section 6.7.2
<i>VDMBusyCounter</i>	<i>nBusyCount</i>	Section 6.7.6

6.8 Reset

Resets are a necessary response to protocol or other error conditions. USB Power Delivery defines four different types of reset:

- *Soft Reset*, which resets protocol.
- *Data Reset* which resets the *USB Communications*.
- *Hard Reset* which resets both the power supplies and protocol
- *Cable Reset* which resets the cable.

6.8.1 Soft Reset and Protocol Error

A **Soft_Reset** Message is used to cause a *Soft Reset* of protocol communication when this has broken down in some way. It **Shall Not** have any impact on power supply operation but is used to correct a *Protocol Error* occurring during an *Atomic Message Sequence* (AMS). The *Soft Reset* **May** be triggered by either *Port Partner* in response to the *Protocol Error*.

Protocol Errors are any unexpected *Message* during an AMS. If the first *Message* in an AMS has been passed to the *Protocol Layer* by the *Policy Engine* but has not yet been sent (i.e., a **GoodCRC** Message acknowledging the *Message* has not been received) when the *Protocol Error* occurs, the *Policy Engine* **Shall Not** issue a *Soft Reset* but **Shall** return to the **PE_SNK_Ready** or **PE_SRC_Ready** state and then process the incoming *Message*. If the incoming *Message* is an *Unexpected Message* received in the **PE_SNK_Ready** or **PE_SRC_Ready** state, the *Policy Engine* **Shall** issue a *Soft Reset*. If the *Protocol Error* occurs during an AMS this **Shall** lead to a *Soft Reset* in order to re-synchronize the *Policy Engine* state machines (see [Section 8.3.3.4, "SOP Soft Reset and Protocol Error State Diagrams"](#)) except when the voltage is transition when a *Protocol Error* **Shall** lead to a *Hard Reset* (see [Section 6.6.11.4, "tProtErrHardReset"](#) and [Section 8.3.3.2, "Policy Engine Source Port State Diagram"](#)). Details of AMS's can be found in [Section 8.3.2.1.3, "Atomic Message Sequences"](#).

An *Unrecognized Message* or *Unsupported Message* received in the **PE_SNK_Ready** or **PE_SRC_Ready** states, **Shall Not** cause a **Soft_Reset** Message to be generated but instead a **Not_Supported** Message **Shall** be generated.

A **Soft_Reset** Message **Shall** be sent regardless of the R_p value either **SinkTxOK** or **SinkTxNG** if it is the correct response in that state.

Note: This means that a **Soft_Reset** Message can be sent during an AMS regardless of the R_p value either **SinkTxOK** or **SinkTxNG** when responding to a *Protocol Error*.

[Table 6.72, "Response to an incoming Message \(except VDM\)"](#) and [Table 6.73, "Response to an incoming VDM"](#) summarize the responses that **Shall** be made to an incoming *Message* including *VDMs*.

Table 6.72 Response to an incoming Message (except VDM)

Recipient's Power Role	Recipient's state	Incoming Message			
		Recognized			Unrecognized
		Supported		Unsupported	
Expected	Unexpected				
Source	PE_SRC_Ready	Process Message	Soft_Reset Message ²	Not_Supported Message ³	Not_Supported Message ³ (except for VDM) See Section 6.4.4.1 for UVDM. See Section 6.4.4.1 for SVDM
	During AMS (power not transitioning ¹)	Process Message	Soft_Reset Message ²		
	During AMS (power transitioning ¹)	Process Message	Hard_Reset Signaling		
Sink	PE_SNK_Ready	Process Message	Soft_Reset Message ²	Not_Supported Message ³	Not_Supported Message ³ (except for VDM) See Section 6.4.4.1 for UVDM. See Section 6.4.4.1 for SVDM
	During AMS (not power transitioned)	Process Message	Soft_Reset Message ²		
	During AMS (power transitioned)	Process Message	Hard_Reset Signaling		

1)

“Power transitioning” means the Policy Engine is in PE_SRC_Transition_Supply State or PE_SNK_Transition_Sink State or PE_FRS_SNK_SRC_Start_AMS State.

2)

The Soft_Reset Message **Shall** be sent using the SOP* of the incoming Message.

3)

The Not_Supported Message **Shall** be sent using the SOP* of the incoming Message.

Table 6.73 Response to an incoming VDM

Recipient's Role	Unstructured VDM			Structured VDM		
	Supported	Unsupported	Unrecognized	Supported	Unsupported	Unrecognized
DFP or UFP	Defined by vendor	Not_Supported Message	Not_Supported Message	See Section 6.13.5	Not_Supported Message	NAK Command
Cable Plug	Defined by vendor	Message Ignored	Message Ignored	See Section 6.13.5	Message Ignored	NAK Command

A failure to see a **GoodCRC** Message in response to any *Message* within *tReceive* (after *nRetryCount* retries), when a *Port Pair* is *Connected*, is indicative of a communications failure resulting in a *Soft Reset* (see [Section 6.6.9.1, "tSoftReset"](#)).

A *Soft Reset* **Shall** impact the USB Power Delivery layers in the following ways:

- *PHY Layer*: Reset not required since the *PHY Layer* resets on each *Packet* transmission/reception.
- *Protocol Layer*: Reset **MessageIDCounter**, **RetryCounter** and state machines.

- *Policy Engine*: Reset state dependent behavior by performing an *Explicit Contract Negotiation*.
- Power supply: **Shall Not** change.

Note: When in *SPR Mode* the *Source* sends a *Source_Capabilities* Message and when in *EPR Mode* the *Source* sends an *EPR_Source_Capabilities* Message.

A *Soft Reset* is performed using an *AMS* (see [Table 8.8, "AMS: Soft Reset"](#)). Message numbers **Shall** be set to zero prior to sending the *Soft_Reset/Accept* Message since the issue might be with the counters. The sender of a *Soft_Reset* Message **Shall** reset its *MessageIDCounter* and *RetryCounter*, the receiver of the Message **Shall** reset its *MessageIDCounter* and *RetryCounter* before sending the *Accept* Message response. Any failure in the *Soft Reset* process will trigger a *Hard Reset* when *SOP Packets* are being used or *Cable Reset*, sent by the *DFP* only, for any other *SOP* Packets*; for example a *GoodCRC* Message is not received during the *Soft Reset* process (see [Section 6.8.3, "Hard Reset"](#) and [Section 6.8.4, "Cable Reset"](#)).

6.8.2 Data Reset

A *Data_Reset* Message is used by a *Port* to reset its USB data connection and to exit all *Alternate Modes* both with its *Port Partner* and in the *Cable Plug(s)*.

- The *Data Reset* process **May** be initiated by either *Port Partner* sending a *Data_Reset* Message.

A *Data Reset* impacts USB Power Delivery in the following ways:

- **Shall Not** change the *Port Power Roles* (*Source/Sink*) or *Port Data Roles* (*DFP/UFP*).
- **Shall Not** change the existing *Explicit Contract*.
- **Shall** cause all *Active Modes* to be exited.
- **Shall** reset the cable by Power cycling *VCONN*.
- The *DFP Shall* become the *VCONN Source*.
- If the *Data Reset* process fails, then the *Port Shall* enter the *ErrorRecovery* State as defined in [\[USB Type-C 2.4\]](#).

See [Section 6.3.14, "Data_Reset Message"](#) for details of *Data Reset* operation.

6.8.3 Hard Reset

Hard Resets are signaled by an ordered set as defined in [Section 5.6.4, "Hard Reset"](#). Both the sender and recipient **Shall** cause their power supplies to return to their default states (see [Section 7.3.3.1, "Source Initiated Hard Reset"](#) and [Section 7.3.3.2, "Sink Initiated Hard Reset"](#) for details of voltage transitions). In addition, their respective *Protocol Layers Shall* be reset as for the *Soft Reset*. This allows the *Attached* devices to be in a state where they can re-establish USB PD communication. *Hard Reset* is retried up to *nHardResetCount* times (see also [Section 6.6.6, "NoResponseTimer"](#) and [Section 6.7.3, "Hard Reset Counter"](#)).

Note: Even though *VBUS* drops to *vSafe0V* during a *Hard Reset* a *Sink* will not see this as a disconnect since this is expected behavior.

A *Hard Reset Shall Not* cause any change to either the R_p/R_d resistor being asserted.

If there has been a *Data Role Swap* the *Hard Reset Shall* cause the *Port Data Role* to be changed back to *DFP* for a *Port* with the R_p resistor asserted and *UFP* for a *Port* with the R_d resistor asserted.

When *VCONN* is supported (see [\[USB Type-C 2.4\]](#)) the *Hard Reset Shall* cause the *Port* with the R_p resistor asserted to supply *VCONN* and the *Port* with the R_d resistor asserted to turn off *VCONN*.

In effect the *Hard Reset* will revert the *Ports* to their default state based on their *CC* line resistors. Removing and reapplying *VCONN* from the *Cable Plugs* also ensures that they re-establish their configuration as either *SOP'* or *SOP''* based on the location of *VCONN* (see [\[USB Type-C 2.4\]](#)).

If the *Hard Reset* is insufficient to clear the error condition, then the *Port* **Shall** use *USB Type-C ErrorRecovery* as defined in [\[USB Type-C 2.4\]](#).

A *Sink* **Shall** be able to send *Hard Reset* Signaling regardless of the value of R_p (see [Section 5.7, "Collision Avoidance"](#)).

6.8.3.1 Cable Plugs and Hard Reset

Cable Plugs **Shall Not** generate *Hard Reset* Signaling but **Shall** monitor for *Hard Reset* Signaling between the *Port Partners* and **Shall** reset when this is detected (see [Section 8.3.3.25.2.2, "Cable Plug Hard Reset State Diagram"](#)). The *Cable Plugs* **Shall** perform the equivalent of a power cycle returning to their initial power up state. This allows the *Port Partners* to be in a state where they can re-establish USB PD communication.

6.8.3.2 Modal Operation and Hard Reset

A *Hard Reset* **Shall** cause *EPR Mode* and all *Active Modes* to be exited by both *Port Partners* and any *Cable Plugs* (see [Section 6.4.4.3.4, "Enter Mode Command"](#)).

6.8.4 Cable Reset

Cable Resets are signaled by an ordered set as defined in [Section 5.6.5, "Cable Reset"](#). Both the sender and recipient of *Cable Reset* Signaling **Shall** reset their respective *Protocol Layers*. The *Cable Plugs* **Shall** perform the equivalent of a power cycle returning to their initial power up state. This allows the *Port Partners* to be in a state where they can re-establish USB PD communication.

The *DFP* must be supplying *VCONN* prior to a *Cable Reset*. If *VCONN* has been turned off the *DFP* **Shall** turn on *VCONN* prior to generating *Cable Reset* Signaling. If there has been a *VCONN Swap* and the *UFP* is currently supplying *VCONN*, the *DFP* **Shall** perform a *VCONN Swap* such that it is supplying *VCONN* prior to generating *Cable Reset* Signaling.

Only a *DFP* **Shall** generate *Cable Reset* Signaling. A *DFP* **Shall** only generate *Cable Reset* Signaling within an *Explicit Contract*.

A *Cable Reset* **Shall** cause all *Active Modes* in the *Cable Plugs* to be exited (see [Section 6.4.4.3.4, "Enter Mode Command"](#)).

6.9 Accept, Reject and Wait

The recipient of a *Request*, *EPR_Request*, *PR_Swap*, *DR_Swap*, *VCONN_Swap*, or *Enter_USB Message* **shall** respond by sending one of the following responses:

- An *Accept Message* in response to a **Valid** request which can be serviced immediately (see [Section 6.3.3, "Accept Message"](#)).
- A *Wait Message* in response to a **Valid** request which cannot be serviced immediately but could be serviced at a later time (see [Section 6.3.12, "Wait Message"](#)).
- A *Reject Message* in response to an **Invalid** request or a request which is outside of the device's design Capabilities (see [Section 6.3.4, "Reject Message"](#)).

6.10 Collision Avoidance

To avoid *Message* collisions due to asynchronous *Messaging* sent from the *Sink*, the *Source* sets R_p to **SinkTxOK** to indicate to the *Sink* that it is OK to initiate an *AMS*. When the *Source* wishes to initiate an *AMS*, it sets R_p to **SinkTxNG**. When the *Sink* detects that R_p is set to **SinkTxOK** it **May** initiate an *AMS*. When the *Sink* detects that R_p is set to **SinkTxNG** it **Shall Not** initiate an *AMS* and **Shall** only send *Messages* that are part of an *AMS* the *Source* has initiated.

Note: This restriction applies to *SOP* AMS*'s i.e., for both *Port to Port* and *Port to Cable Plug* communications.

If a transition into the **PE_SRC_Ready** state will result in an immediate transition out of the **PE_SRC_Ready** state within **tSrcHoldsBus** e.g. it is due to a *Protocol Error* that has not resulted in a *Soft Reset*, then the notifications of the end of *AMS* and first *Message* in an *AMS* **May** Not be sent to avoid changing the R_p value unnecessarily.

Note: A *Sink* can still send **Hard Reset** Signaling at any time.

6.11 Message Discarding

On receiving a received *Message* on *SOP*, the *Protocol Layer* **Shall Discard** any pending *SOP** *Messages*. A received *Message* on *SOP'/SOP''* **Shall Not** cause any pending *SOP** *Messages* to be **Discarded**.

It is assumed that *Messages* using *SOP'/SOP''* constitute a simple request/response *AMS*, with the *Cable Plug* providing the response so there is no reason for a pending *SOP** *Message* to be **Discarded**. There can only be one *AMS* between the *Port Partners*, and these also take priority over *Cable Plug* communications so a *Message* received on *SOP* will always cause a *Message* pending on *SOP** to be **Discarded**.

[Table 6.74, "Message Discarding"](#) for details of the *Messages* that **Shall/ Shall Not** be **Discarded**.

Table 6.74 Message Discarding

Message pending transmission	Message received	Message to be Discarded
<i>SOP</i>	<i>SOP</i>	Outgoing <i>Message</i>
<i>SOP</i>	<i>SOP'/SOP''</i>	Incoming <i>Message</i>
<i>SOP'</i>	<i>SOP</i>	Outgoing <i>Message</i>
<i>SOP'</i>	<i>SOP'</i>	Incoming <i>Message</i>
<i>SOP'</i>	<i>SOP''</i>	Incoming <i>Message</i>
<i>SOP''</i>	<i>SOP</i>	Outgoing <i>Message</i>
<i>SOP''</i>	<i>SOP'</i>	Incoming <i>Message</i>
<i>SOP''</i>	<i>SOP''</i>	Incoming <i>Message</i>

6.12 State behavior

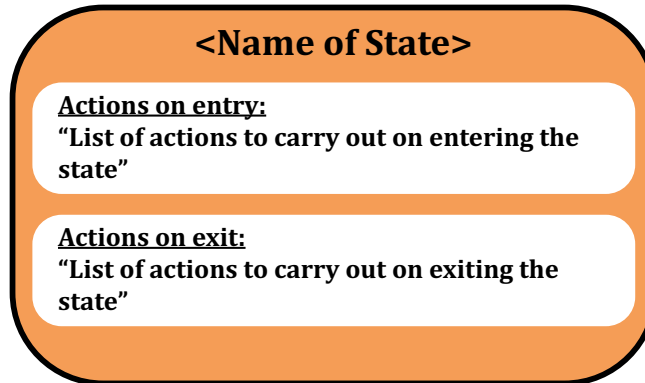
6.12.1 Introduction to state diagrams used in Chapter 6

The state diagrams defined in [Section 6.12, "State behavior"](#) are **Normative** and **Shall** define the operation of the Power Delivery Protocol Layer.

Note: These state diagrams are not intended to replace a well written and robust design.

[Figure 6.57, "Outline of States"](#) shows an outline of the states defined in the following sections. At the top there is the name of the state. This is followed by "Actions on entry" a list of actions carried out on entering the state and in some states "Actions on exit" a list of actions carried out on exiting the state.

Figure 6.57 Outline of States

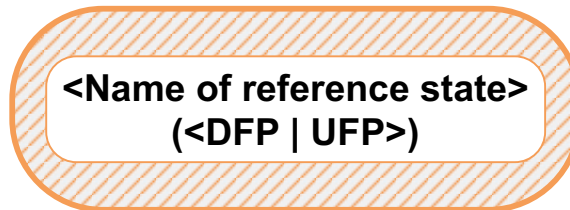


Transitions from one state to another are indicated by arrows with the conditions listed on the arrow. Where there are multiple conditions, these are connected using either a logical OR "|" or a logical AND "&." The inverse of a condition is shown with a "NOT" in front of the condition.

In some cases, there are transitions which can occur from any state to a particular state. These are indicated by an arrow which is unconnected to a state at one end, but with the other end (the point) connected to the final state.

In some state diagrams it is necessary to enter or exit from states in other diagrams. [Figure 6.57, "Outline of States"](#) indicates how such references are made. The reference is indicated with a hatched box. The box contains the name of the referenced state.

Figure 6.58 References to states



Timers are included in many of the states. Timers are initialized (set to their starting condition) and run (timer is counting) in the state it is referenced. As soon as the state is exited then the timer is no longer active. Timeouts of the timers are listed as conditions on state transitions.

Conditions listed on state transitions will come from one of three sources:

- Messages received from the *PHY Layer*.
- Events triggered within the *Protocol Layer* e.g., timer timeouts
- Message and related indications passed up to the *Policy Engine* from the *Protocol Layer* (Message sent; Message received etc.)

6.12.2 State Operation

The following section details *Protocol Layer* State Operation when sending and receiving *SOP* Packets*.

For each *SOP' Communication* being sent and received there **shall** be separate *Protocol Layer* Transmission and *Protocol Layer* Reception and *Hard Reset* State Machine instances, with their own counter and timer instances. When *Chunking* is supported there **shall** be separate *Chunked Tx*, *Chunked Rx*, and *Chunked Message Router* State Machine instances.

Soft Reset **shall** only apply to the State Machine instances it is targeted at based on the type of *SOP* Packet* used to send the **Soft_Reset** Message. The *Hard Reset* State Machine (including *Cable Reset*) **shall** apply simultaneously to all *Protocol Layer* State Machine instances active in the *DFP*, *UFP* and *Cable Plug* (if present).

6.12.2.1 Protocol Layer Chunking

6.12.2.1.1 Architecture of Device Including Chunking Layer

The *Chunking* component resides in the *Protocol Layer* between the *Policy Engine* and *Protocol Tx/Rx*. [Figure 6.59, "Chunking architecture Showing Message and Control Flow"](#) illustrates the relationship between components.

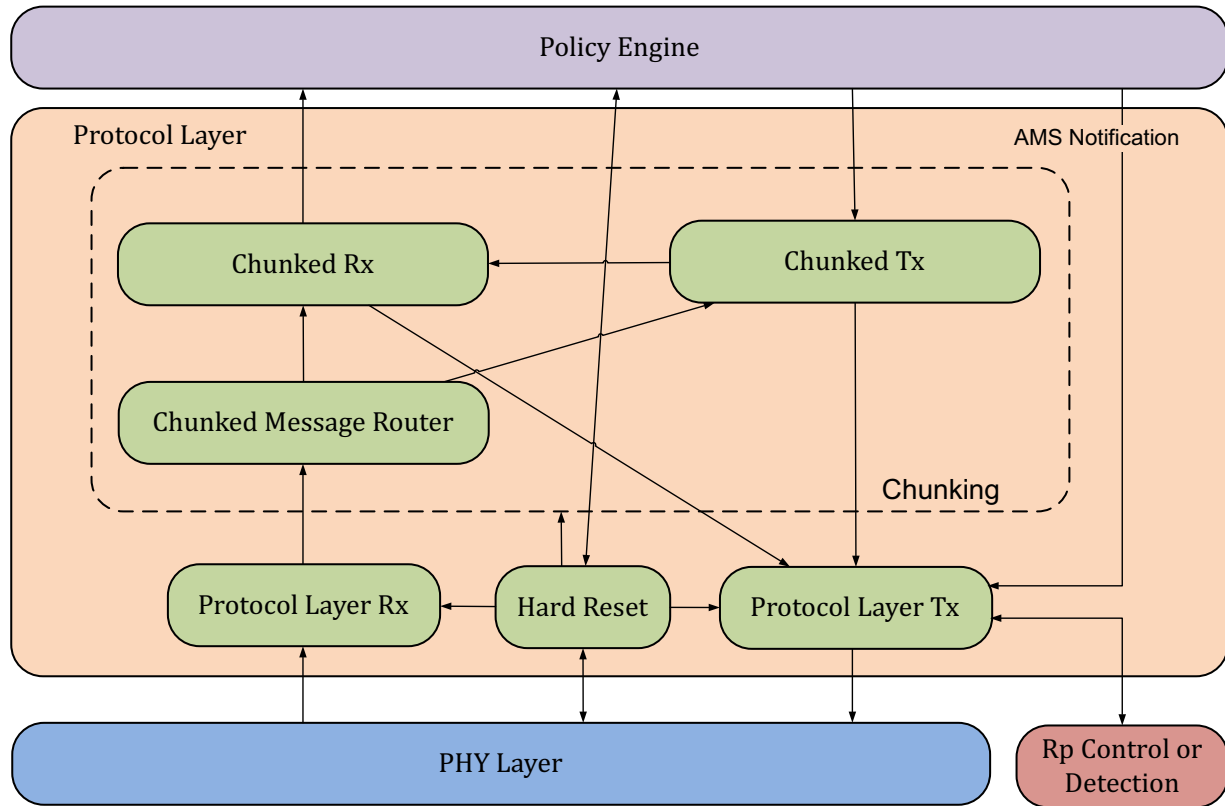
The *Chunking Layer* comprises three related state machines:

- *Chunked Rx*.
- *Chunked Tx*.
- *Chunked Message Router*.

Note: The consequence of this architecture is that the *Policy Engine* deals entirely in *Unchunked Messages*. It will not receive (and might not respond to) a *Message* until all the related chunks have been collated.

If a PD device or *Cable Plug* has no requirement to handle any *Message* requiring more than one *Chunk* of any *Extended Message*, it **may** omit the *Chunking Layer*. In this case it **shall** implement the **ChunkingNotSupportedTimer** to ensure compatible operation with partners which support *Chunking* (see [Section 6.6.18.1, "ChunkingNotSupportedTimer"](#) and [Section 8.3.3.6, "Not Supported Message State Diagrams"](#)).

Figure 6.59 Chunking architecture Showing Message and Control Flow



6.12.2.1.1.1 Optional Abort Mechanism

Long *Chunked Messages* bring with them the potential problem that they could prevent urgent *Messages* from being transmitted in a timely manner. An **Optional** Abort mechanism is provided to remedy this problem.

The Abort Flag referred to in the diagrams below **May** be set and examined by the *Policy Engine*. The specific means are left to the implementer.

6.12.2.1.1.2 Aborting Sending a Long-Chunked Message

A long-*Chunked Message* being sent **May** be aborted by setting the **Optional** Abort Flag. The *Message* **Shall** be considered aborted when the Abort Flag is again cleared by the *Chunked Tx* state machine.

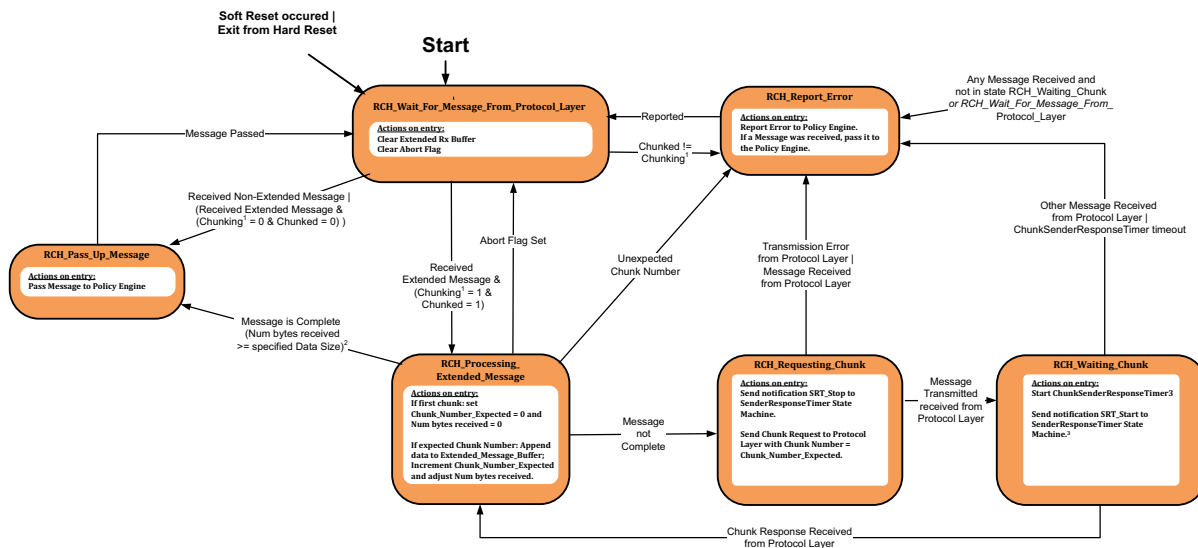
6.12.2.1.1.3 Aborting Receiving a Long-Chunked Message

If the **Optional** Abort mechanism has been implemented, any *Message* sent while a *Chunked Message* receive is in progress will result in an error report being received by the *Policy Engine*, to indicate that the *Message* request has been **Discarded**. If the *Message* was urgent the *Policy Engine* might set the Abort Flag, which will result in the incoming *Chunked Message* being aborted. The Abort Flag being cleared by the *Chunked Rx* state machine indicates that the urgent *Message* can now be sent.

6.12.2.1.2 Chunked Rx State Diagram

[Figure 6.60, "Chunked Rx State Diagram"](#) shows the state behavior for the *Chunked Rx* State Machine. This recognizes whether *Chunked* received *Messages* are involved and deals with requesting chunks when they are. It also performs validity checks on all *Messages* related to *Chunking*.

Figure 6.60 Chunked Rx State Diagram



- 1) Chunking is an internal state that is set to 1 if the 'Unchunked Extended Messages Supported' bit in either Source Capabilities or Request is 0. It defaults to 1 and is set after the first exchange of Source Capabilities and Request. It is also set to 1 for *SOP*² or *SOP*³ communication.
- 2) Additional bytes received over specified *Data Size* will be because of padding in the last chunk.
- 3) This state is responsible for starting two timers of similar length. The implementor *Should* mitigate against more than one of these timers resulting in recovery action.

6.12.2.1.2.1 RCH_Wait_For_Message_From_Protocol_Layer State

The *Chunked* Rx State Machine **Shall** enter the *RCH_Wait_For_Message_From_Protocol_Layer* state:

- At startup.
- As a result of a *Soft Reset* occurring.
- On exit from a *Hard Reset*.

On entry to the *RCH_Wait_For_Message_From_Protocol_Layer* state the *Chunked* Rx state machine clears the Extended Rx Buffer and clears the *Optional* Abort Flag.

In the *RCH_Wait_For_Message_From_Protocol_Layer* state the *Chunked* Rx state machine waits until the *Chunked* Message Router passes up a received *Message*.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Pass_Up_Message* state when:

- A non-Extended *Message* is passed up from the *Chunked* Message Router.
- An *Extended Message* is passed up from the *Chunked* Message Router, and the *Policy Engine* has determined that we are not doing *Chunking*, and the *Message* has its *Chunked* bit set to 0b.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Processing_Extended_Message* state when:

- An *Extended Message* is passed up from the *Chunked* Message Router, and the *Policy Engine* has determined that we are doing *Chunking*, and the *Message* has its *Chunked* bit set to 1b.

6.12.2.1.2.2 RCH_Pass_Up_Message State

On entry to the *RCH_Pass_Up_Message* state the *Chunked* Rx state machine **Shall** pass the received *Message* to the *Policy Engine*.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Wait_For_Message_From_Protocol_Layer* state when:

- The *Message* has been passed.

6.12.2.1.2.3 RCH_Processing_Extended_Message State

On entry to the *RCH_Processing_Extended_Message* state the *Chunked* Rx state machine **Shall**:

- If this is the first chunk:
 - Set *Chunk_Number_Expected* = 0.
 - Set Num bytes received = 0.
- If chunk contains the expected *Chunk Number*:
 - Append its data to the *Extended_Message_Buffer*.
 - Increment *Chunk_Number_Expected*.
 - Adjust Num bytes received.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Pass_Up_Message* state when:

- The *Message* is complete (i.e., Num bytes received \geq specified *Data Size*).

Note: The inequality allows for padding bytes in the last chunk, which are not actually part of the *Extended Message*).

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Requesting_Chunk* state when:

- The *Message* is not yet complete.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Report_Error* state when:

- An unexpected *Chunk Number* is received.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Wait_For_Message_From_Protocol_Layer* state when:

- The *Optional* Abort Flag is set.

6.12.2.1.2.4 RCH_Requesting_Chunk State

On entry to the *RCH_Requesting_Chunk* state the *Chunked* Rx state machine **Shall**:

- Send notification SRT_Stop to *SenderResponseTimer* state machine (see [Section 8.3.3.1.1, "SenderResponseTimer State Diagram"](#)).
- Send *Chunk Request to Protocol Layer* with *Chunk Number* = *Chunk_Number_Expected*.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Waiting_Chunk* state when:

- *Message Transmitted* is received from the *Protocol Layer*.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Report_Error* state when:

- Transmission Error is received from the *Protocol Layer*, or
- A *Message* is received from the *Protocol Layer*.

6.12.2.1.2.5 RCH_Waiting_Chunk State

On entry to the *RCH_Waiting_Chunk* state the *Chunked* Rx state machine **Shall**:

- Start the *ChunkSenderResponseTimer*.
- Send notification SRT_Start to *SenderResponseTimer* state machine (see [Section 8.3.3.1.1, "SenderResponseTimer State Diagram"](#)).

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Processing_Extended_Message* state when:

- A *Chunk* is received from the *Protocol Layer*.

The *Chunked* Rx State Machine **Shall** transition to the *RCH_Report_Error* state when:

- A *Message*, other than a *Chunk*, is received from the *Protocol Layer*, or
- The *ChunkSenderResponseTimer* expires.

6.12.2.1.2.6 RCH_Report_Error State

The *Chunked* Rx State Machine **Shall** enter the *RCH_Report_Error* state:

- When any *Message* is received and the *Chunked* Rx State Machine is not in one of the states *RCH_Waiting_Chunk* or *RCH_Wait_For_Message_From_Protocol_Layer*.

On entry to the *RCH_Report_Error* state the *Chunked* Rx state machine **Shall**:

- Report the error to the *Policy Engine*.
- If the state was entered because a *Message* was received, this *Message* **Shall** be passed to the *Policy Engine*.

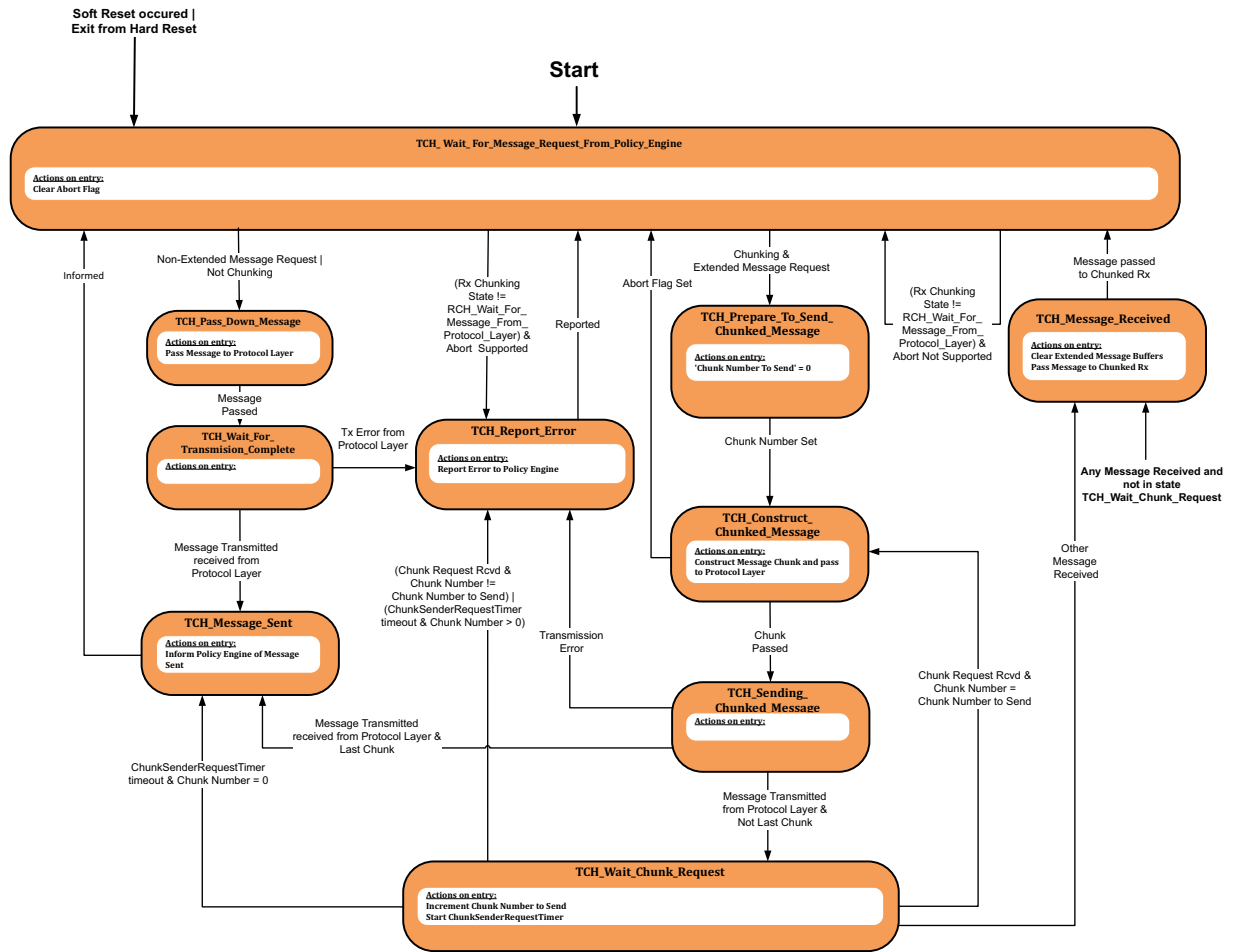
The *Chunked* Rx State Machine **Shall** transition to the *RCH_Wait_For_Message_From_Protocol_Layer* state when:

- The error has been reported.
- Any *Message* received was passed to the *Policy Engine*.

6.12.2.1.3 Chunked Tx State Diagram

Figure 6.61, "Chunked Tx State Diagram" shows the state behavior for the *Chunked Tx State Machine*. This recognizes whether *Chunked* transmitted *Messages* are involved and deals with sending chunks and waiting for chunk requests when they are. It also performs validity checks on all related *Messages* related to *Chunking*.

Figure 6.61 Chunked Tx State Diagram



6.12.2.1.3.1 TCH_Wait_For_Message_Request_From_Policy_Engine State

The *Chunked Tx State Machine* **shall** enter the *TCH_Wait_For_Message_Request_From_Policy_Engine* state:

- At startup.
- As a result of a *Soft Reset* occurring.
- On exit from a *Hard Reset*.

On entry to the *TCH_Wait_For_Message_Request_From_Policy_Engine* state the *Chunked Tx* state machine clears the **Optional** Abort Flag.

In the *TCH_Wait_For_Message_Request_From_Policy_Engine* state the *Chunked Tx* State Machine waits until the *Policy Engine* sends it a *Message Request*.

The *Chunked Tx* State Machine **shall** transition to the *TCH_Pass_Down_Message* state when:

- A non-Extended *Message Request* is received from the *Policy Engine*, or
- A *Message Request* is received from the *Policy Engine* and the link is not *Chunking*.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Prepare_To_Send_Chunked_Message* state when:

- An *Extended Message* Request is received from the *Policy Engine*, and the link is *Chunking*.

The *Chunked* Tx State Machine **Shall Discard** the *Message* Request and remain in the *TCH_Wait_For_Message_Request_From_Policy_Engine* state when:

- The *Chunked* Rx state is any other than *RCH_Wait_For_Message_From_Protocol_Layer*, and the **Optional** Abort Flag has not been implemented.

The *Chunked* Tx State Machine **Shall Discard** the *Message* Request and enter the *TCH_Report_Error* state when:

- The *Chunked* Rx state is any other than *RCH_Wait_For_Message_From_Protocol_Layer* and the **Optional** Abort Flag has been implemented.

6.12.2.1.3.2 TCH_Pass_Down_Message State

On entry to the *TCH_Pass_Down_Message* state the *Chunked* Tx State Machine **Shall** pass the *Message* to the *Protocol Layer*.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Wait_For_Transmission_Complete* state when:

- The *Message* has been passed to the *Protocol Layer*.

6.12.2.1.3.3 TCH_Wait_For_Transmission_Complete State

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Message_Sent* state when:

- *Message* Transmitted has been received from the *Protocol Layer*.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Report_Error* state when:

- Transmission Error has been received from the *Protocol Layer*.

6.12.2.1.3.4 TCH_Message_Sent State

On entry to the *TCH_Message_Sent* state the *Chunked* Tx State Machine **Shall**:

- Inform the *Policy Engine* that the *Message* has been sent.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Wait_For_Message_Request_From_Policy_Engine* state when:

- The *Policy Engine* has been informed.

6.12.2.1.3.5 TCH_Prepare_To_Send_Chunked_Message State

On entry to the *TCH_Prepare_To_Send_Chunked_Message* state the *Chunked* Tx State Machine **Shall**:

- Set 'Chunk Number To Send' to zero.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Construct_Chunked_Message* state when:

- 'Chunk Number To Send' has been set to zero.

6.12.2.1.3.6 TCH_Construct_Chunked_Message State

On entry to the *TCH_Construct_Chunked_Message* state the *Chunked* Tx State Machine **Shall**:

- Construct a *Message Chunk* and pass it to the *Protocol Layer*.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Sending_Chunked_Message* state when:

- The *Message Chunk* has been passed to the *Protocol Layer*.

The *Chunked* Tx State Machine **Shall** transition to the *TCH_Wait_For_Message_Request_From_Policy_Engine* state when:

- The **Optional** Abort Flag is set.

6.12.2.1.3.7 TCH_Sending_Chunked_Message State

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Wait_Chunk_Request** state when:

- *Message* Transmitted is received from *Protocol Layer* and this was not the last chunk.

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Message_Sent** state when:

- *Message* Transmitted is received from *Protocol Layer* and this was the last chunk.

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Report_Error** state when:

- Transmission Error has been received from the *Protocol Layer*.

6.12.2.1.3.8 TCH_Wait_Chunk_Request State

On entry to the **TCH_Wait_Chunk_Request** state the *Chunked* Tx State Machine **Shall**:

- Increment *Chunk Number* to Send.
- Start **ChunkSenderRequestTimer**.

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Report_Error** state when:

- A *Chunk* Request has been received and the **Chunk Number** does not equal *Chunk Number* to Send or
- **ChunkSenderRequestTimer** has expired and **Chunk Number** is greater than zero.

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Message_Sent** state when:

- **ChunkSenderRequestTimer** has expired and **Chunk Number** equals zero.

Note: This is the mechanism which allows the remote *Port Partner* or *Cable Plug* to omit the *Chunking Layer*. The *Policy Engine* will receive a *Message* Sent signal if the remote *Port Partner* or *Cable Plug* is present (**GoodCRC** *Message* received) but does not send a *Chunk* Request. After this the remote *Port Partner* will send a **Not_Supported** *Message*, or the *Cable Plug* will **Ignore** the *Chunked Message*.

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Message_Received** state when:

- Any other *Message* than *Chunk* Request is received.

6.12.2.1.3.9 TCH_Message_Received State

The *Chunked* Tx State Machine **Shall** enter the **TCH_Message_Received** state:

- When any *Message* is received, and the *Chunked* Tx State Machine is not in the **TCH_Wait_Chunk_Request** state.

On entry to the **TCH_Message_Received** state the *Chunked* Tx State Machine **Shall**:

- Clear the *Extended Message* Buffers.
- Pass the received *Message* to *Chunked Rx Engine*.

The *Chunked* Tx State Machine **Shall** transition to the **TCH_Wait_For_Message_Request_From_Policy_Engine** state when:

- The received *Message* has been passed to the *Chunked Rx Engine*.

6.12.2.1.3.10 TCH_Report_Error State

On entry to the **TCH_Report_Error** state the *Chunked* Tx State Machine **Shall**:

- Report the error to the *Policy Engine*.

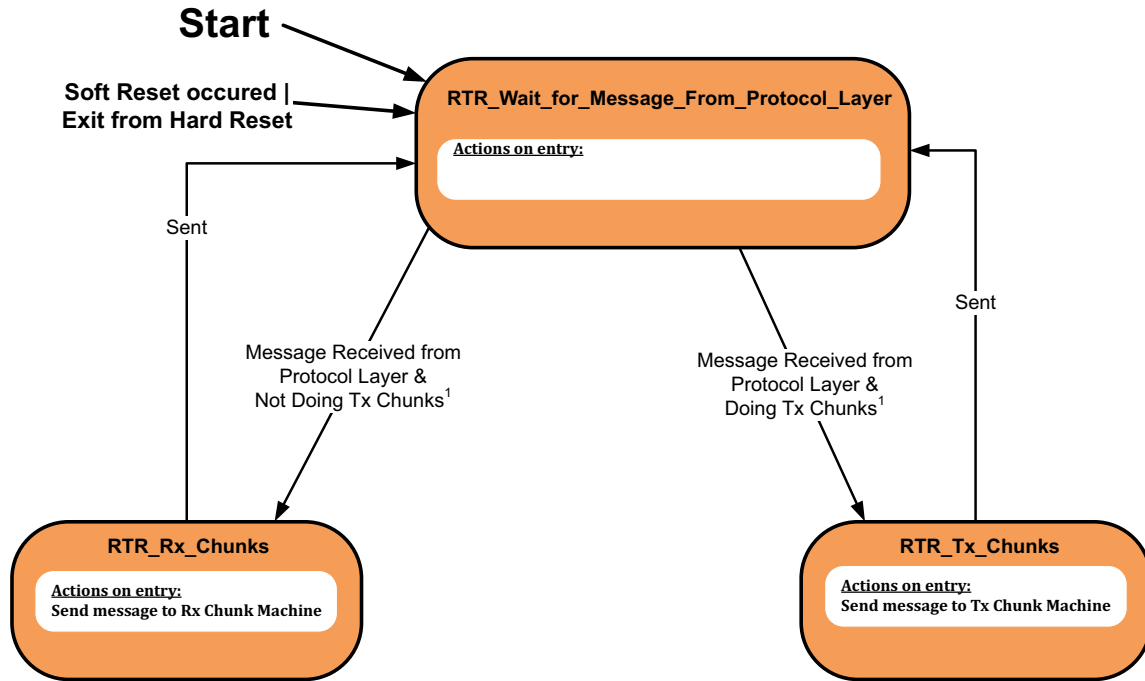
The *Chunked Tx State Machine* **Shall** transition to the *TCH_Wait_For_Message_Request_From_Policy_Engine* state when:

- The error has been reported.

6.12.2.1.4 Chunked Message Router State Diagram

Figure 6.62, "*Chunked Message Router State Diagram*" shows the state behavior for the *Chunked Message Router*. This determines to which state machine an incoming *Message* is routed to (*Chunked Rx*, *Chunked Tx* or direct to *Policy Engine*).

Figure 6.62 Chunked Message Router State Diagram



- 1) Doing Tx Chunks means that *Chunked Tx State Machine* is not in the *TCH_Wait_For_Message_Request_From_Policy_Engine* state.
 2) Messages are taken to include notification about transmission success or otherwise of Messages.

6.12.2.1.4.1 RTR_Wait_for_Message_From_Protocol_Layer State

In the *RTR_Wait_for_Message_From_Protocol_Layer* state the *Chunked Message Router* waits until the *Protocol Layer* sends it a received *Message*.

The *Chunked Message Router* **Shall** transition to the *RTR_Rx_Chunks* state when:

- A *Message* is received from the *Protocol Layer*, and the combined *Chunking* is not doing *Tx Chunks*.

The *Chunked Message Router* **Shall** transition to the *RTR_Tx_Chunks* state when:

- A *Message* is received from the *Protocol Layer*, and the combined *Chunking* is doing *Tx Chunks*.

6.12.2.1.4.2 RTR_Rx_Chunks State

On entry to the *RTR_Rx_Chunks* state the *Chunked Message Router* **Shall**:

- Send the *Message* to the *Chunked Rx State Machine*.
- Transition to the *RTR_Wait_for_Message_From_Protocol_Layer* state.

On entry to the *RTR_Tx_Chunks* state the *Chunked* Message Router **Shall**:

- Send the *Message* to the *Chunked* Tx State Machine.
- Transition to the *RTR_Wait_for_Message_From_Protocol_Layer* state.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Construct_Message* state when:

- A *Message* request is received from the *Policy Engine* which is not a *Soft_Reset* *Message*.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Layer_Reset_for_Transmit* state when:

- A *Message* request is received from the *Policy Engine* which is a *Soft_Reset* *Message*.

6.12.2.2.1.3 PRL_Tx_Layer_Reset_for_Transmit State

On entry to the *PRL_Tx_Layer_Reset_for_Transmit* state the *Protocol Layer* **Shall** reset the *MessageIDCounter*. The *Protocol Layer* **Shall** transition *Protocol Layer Message* reception to the *PRL_Rx_Wait_for_PHY_Message* state (see [Section 6.12.2.3.1, "PRL_Rx_Wait_for_PHY_Message state"](#)) in order to reset the stored *MessageID*.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Construct_Message* state when:

- The layer reset actions in this state have been completed.

6.12.2.2.1.4 PRL_Tx_Construct_Message State

On entry to the *PRL_Tx_Construct_Message* state the *Protocol Layer* **Shall** construct the *Message* requested by the *Policy Engine*, or resend a previously constructed *Message*, and then pass this *Message* to the *PHY Layer*.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Wait_for_PHY_Response* state when:

- The *Message* has been sent to the *PHY Layer*.

6.12.2.2.1.5 PRL_Tx_Wait_for_PHY_Response State

On entry to the *PRL_Tx_Wait_for_PHY_Response* state, once the *Message* has been sent, the *Protocol Layer* **Shall** initialize and run the *CRCReceiveTimer* (see [Section 6.6.1, "CRCReceiveTimer"](#)).

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Match_MessageID* state when:

- A *GoodCRC* *Message* response is received from the *PHY Layer*.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Check_RetryCounter* state when:

- The *CRCReceiveTimer* times out.
- Or the *PHY Layer* indicates that a *Message* has been **Discarded** due to the channel being busy but the channel is now *Idle* (see [Section 5.7, "Collision Avoidance"](#)).

6.12.2.2.1.6 PRL_Tx_Match_MessageID State

On entry to the *PRL_Tx_Match_MessageID* state the *Protocol Layer* **Shall** compare the *MessageIDCounter* and the *MessageID* of the received *GoodCRC* *Message*.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Message_Sent* state when:

- The *MessageIDCounter* and the *MessageID* of the received *GoodCRC* *Message* match.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Check_RetryCounter* state when:

- The *MessageIDCounter* and the *MessageID* of the received *GoodCRC* *Message* do not match.

6.12.2.2.1.7 PRL_Tx_Message_Sent State

On entry to the *PRL_Tx_Message_Sent* state the *Protocol Layer* **Shall** increment the *MessageIDCounter* and inform the *Policy Engine* that the *Message* has been sent.

The *Protocol Layer* **Shall** transition to the *PRL_Tx_Wait_for_Message_Request* state when:

- The *Policy Engine* has been informed that the *Message* has been sent.

6.12.2.2.1.8 PRL_Tx_Check_RetryCounter State

On entry to the **PRL_Tx_Check_RetryCounter** state the *Protocol Layer* in a *DFP* or *UFP* **Shall** increment the value of the **RetryCounter** and then check it in order to determine whether it is necessary to retry sending the *Message*.

Note: *Cable Plugs* do not retry *Messages* and so do not use the **RetryCounter**.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Construct_Message** state in order to retry *Message* sending when:

- **RetryCounter** ≤ **nRetryCount** and
- This is not a *Cable Plug* and
- This is an *Extended Message* with **Data Size** ≤ **MaxExtendedMsgLegacyLen** or
- This is an *Extended Message* that has been *Chunked*.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Transmission_Error** state when:

- **RetryCounter** > **nRetryCount** or
- This is a *Cable Plug*, which does not retry.
- This is an *Extended Message* with **Data Size** > **MaxExtendedMsgLegacyLen** that has not been *Chunked*.

6.12.2.2.1.9 PRL_Tx_Transmission_Error State

On entry to the **PRL_Tx_Transmission_Error** state the *Protocol Layer* **Shall** increment the **MessageIDCounter** and inform the *Policy Engine* of the transmission error.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Wait_for_Message_Request** state when:

- The *Policy Engine* has been informed of the transmission error.

6.12.2.2.1.10 PRL_Tx_Discard_Message State

Protocol Layer Message transmission **Shall** enter the **PRL_Tx_Discard_Message** state whenever:

- *Protocol Layer Message* reception receives an incoming *Message* or
- The *Fast Role Swap Request* is being transmitted (see [Section 5.8.5.6, "Fast Role Swap Transmission"](#))
- The *Fast Role Swap Request* is detected (see [Section 5.8.6.3, "Fast Role Swap Detection"](#)).

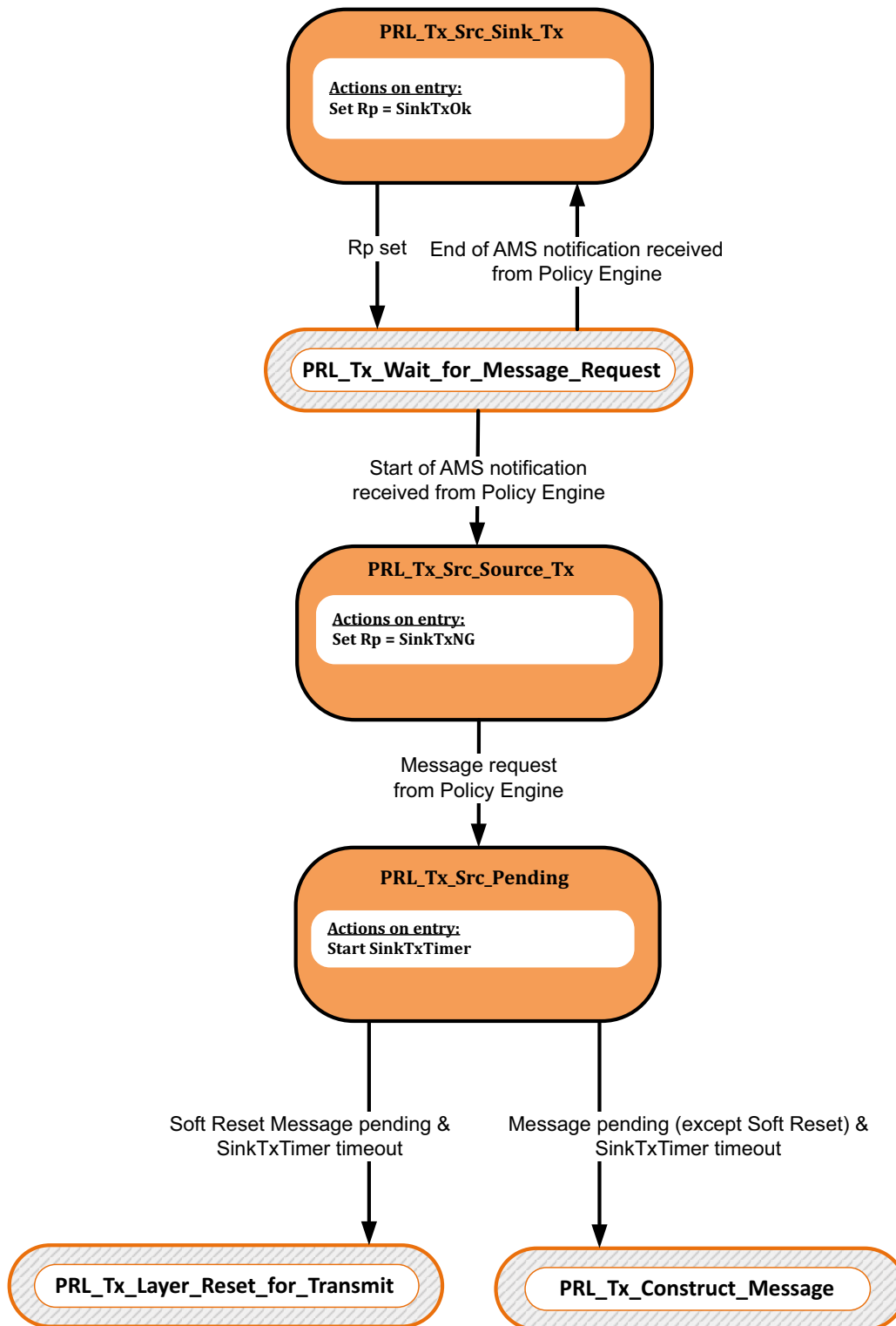
On entry to the **PRL_Tx_Discard_Message** state, if there is a *Message* queued awaiting transmission, the *Protocol Layer* **Shall Discard** the *Message* according to the rules in [Section 6.11, "Message Discarding"](#) and increment the **MessageIDCounter**.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_PHY_Layer_Reset** state when:

- Discarding is complete i.e., the *Message* queue is empty.

Figure 6.64, "Source Protocol Layer Message Transmission State Diagram" shows the state behavior for the *Protocol Layer* in a *Source* when transmitting a *Message*.

Figure 6.64 Source Protocol Layer Message Transmission State Diagram



6.12.2.2.2.1 PRL_Tx_Src_Sink_Tx State

In the **PRL_Tx_Src_Sink_Tx** state the *Source* sets R_p to **SinkTxOK** allowing the *Sink* to start an *Atomic Message Sequence (AMS)*.

The *Protocol Layer* in a *Source* **Shall** transition from the **PRL_Tx_Wait_for_Message_Request** state to the **PRL_Tx_Src_Sink_Tx** state when:

- A notification is received from the *Policy Engine* that the end of an *AMS* has been reached.

On entry to the **PRL_Tx_Src_Sink_Tx** state the *Protocol Layer* **Shall** request the *PHY Layer* to R_p to **SinkTxOK**.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Wait_for_Message_Request** state when:

- R_p has been set.

6.12.2.2.2.2 PRL_Tx_Src_Source_Tx State

In the **PRL_Tx_Src_Source_Tx** state the *Source* sets R_p to **SinkTxNG** allowing the *Source* to start an *Atomic Message Sequence (AMS)*.

The *Protocol Layer* in a *Source* **Shall** transition from the **PRL_Tx_Wait_for_Message_Request** state to the **PRL_Tx_Src_Source_Tx** state when:

- A notification is received from the *Policy Engine* that an *AMS* will be starting.

On entry to the **PRL_Tx_Src_Source_Tx** state the *Protocol Layer* **Shall** set R_p to **SinkTxNG**.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Src_Pending** state when:

- A *Message* request is received from the *Policy Engine*.

6.12.2.2.2.3 PRL_Tx_Src_Pending State

In the **PRL_Tx_Src_Pending** state the *Protocol Layer* has a *Message* buffered ready for transmission.

On entry to the **PRL_Tx_Src_Pending** state the **SinkTxTimer** **Shall** be initialized and run.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Construct_Message** state when:

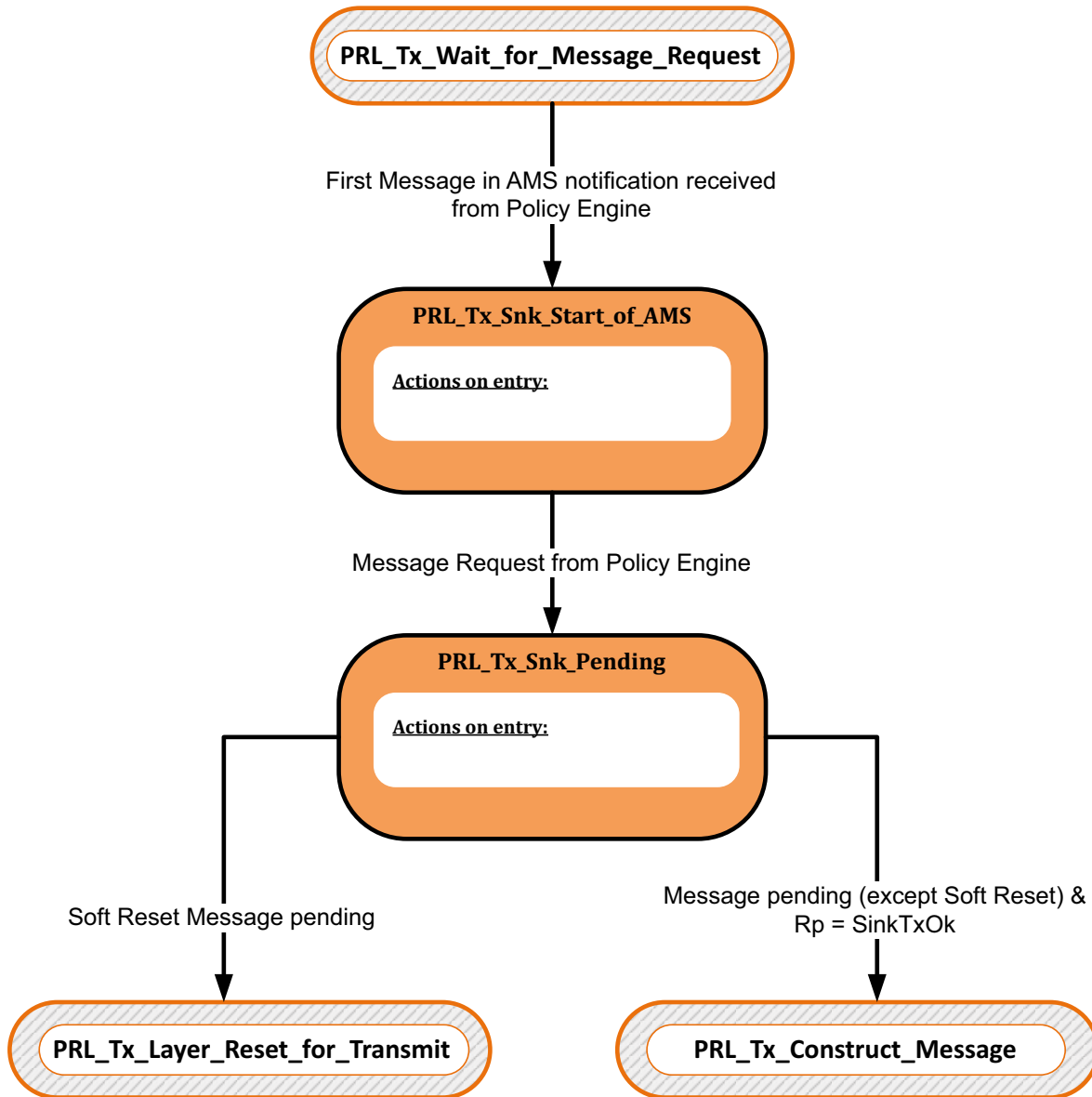
- The pending *Message* request from the *Policy Engine* is not a **Soft_Reset** *Message* and
- The **SinkTxTimer** times out.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Layer_Reset_for_Transmit** state when:

- The pending *Message* request from the *Policy Engine* is a **Soft_Reset** *Message* and
- The **SinkTxTimer** times out.

Figure 6.65, "*Sink Protocol Layer Message Transmission State Diagram*" shows the state behavior for the *Protocol Layer* in a *Sink* when transmitting a *Message*.

Figure 6.65 Sink Protocol Layer Message Transmission State Diagram



6.12.2.2.3.1

PRL_Tx_Snk_Start_of_AMS State

In the **PRL_Tx_Snk_Start_of_AMS** state the *Protocol Layer* waits for the first *Message* in a *Sink* initiated AMS.

The *Protocol Layer* in a *Sink* **shall** transition from the **PRL_Tx_Wait_for_Message_Request** state to the **PRL_Tx_Snk_Start_of_AMS** state when:

- A notification is received from the *Policy Engine* that the next *Message* the *Sink* will send is the start of an AMS.

The *Protocol Layer* **shall** transition to the **PRL_Tx_Snk_Pending** state when:

- A *Message* request is received from the *Policy Engine*.

6.12.2.2.3.2 PRL_Tx_Snk_Pending State

In the **PRL_Tx_Snk_Pending** state the *Protocol Layer* has the first *Message* in a *Sink* initiated *AMS* ready to send and is waiting for R_p to transition to **SinkTxOK** before sending the *Message*.

The *Protocol Layer* **Shall** transition to the **PRL_Tx_Construct_Message** state when:

- A *Message* is Pending that is not a **Soft_Reset** *Message* and
- R_p is set to **SinkTxOK**.

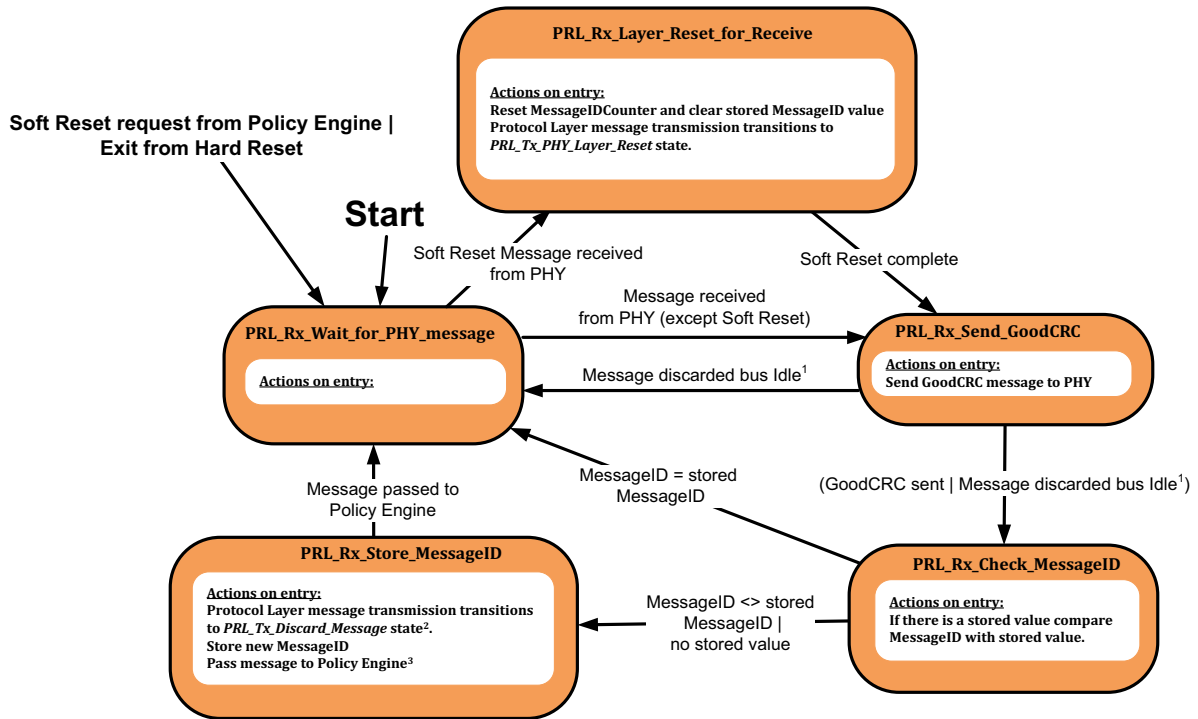
The *Protocol Layer* **Shall** transition to the **PRL_Tx_Layer_Reset_for_Transmit** state when:

- A **Soft_Reset** *Message* is pending.

6.12.2.3 Protocol Layer Message Reception

Figure 6.66, "Protocol layer Message reception" shows the state behavior for the *Protocol Layer* when receiving a Message.

Figure 6.66 Protocol layer Message reception



¹) This indication is sent by the PHY when a message has been **Discarded** due to CC being busy, and after CC becomes idle again (see Section 5.7 "Collision Avoidance"). Two alternate allowable transitions are shown.

²) In the case of a Ping message being received, in order to maintain robust communications in the presence of collisions, the outgoing message **Should Not** be Discarded.

³) See Section 6.11 "Message Discarding" for details of when Messages are discarded.

6.12.2.3.1 PRL_Rx_Wait_for_PHY_Message state

The *Protocol Layer* **Shall** enter the *PRL_Rx_Wait_for_PHY_Message* state:

- At startup.
- As a result of a *Soft Reset* request from the *Policy Engine*.
- On exit from a *Hard Reset*.

In the *PRL_Rx_Wait_for_PHY_Message* state the *Protocol Layer* waits until the *PHY Layer* passes up a received Message.

The *Protocol Layer* **Shall** transition to the *PRL_Rx_Send_GoodCRC* state when:

- A Message is passed up from the *PHY Layer*.

The *Protocol Layer* **Shall** transition to the *PRL_Rx_Layer_Reset_for_Receive* state when:

- A *Soft_Reset* Message is received from the *PHY Layer*.

6.12.2.3.2 PRL_Rx_Layer_Reset_for_Receive state

On entry to the **PRL_Rx_Layer_Reset_for_Receive** state the *Protocol Layer* **Shall** reset the **MessageIDCounter** and clear the stored **MessageID**. The *Protocol Layer* **Shall** transition *Protocol Layer Message* transmission to the **PRL_Tx_Wait_for_Message_Request** state (see [Section 6.12.2.2.1.2, "PRL Tx Wait for Message Request State"](#)).

The *Protocol Layer* **Shall** transition to the **PRL_Rx_Send_GoodCRC** State when:

- The *Soft Reset* actions in this state have been completed.

6.12.2.3.3 PRL_Rx_Send_GoodCRC state

On entry to the **PRL_Rx_Send_GoodCRC** state the *Protocol Layer* **Shall** construct a **GoodCRC** Message and request the *PHY Layer* to transmit it.

The *Protocol Layer* **Shall** transition to the **PRL_Rx_Check_MessageID** state when:

- The **GoodCRC** Message has been passed to the *PHY Layer*.

When the *PHY Layer* indicates that a Message has been **Discarded** due to CC being busy but CC is now *Idle* (see [Section 5.7, "Collision Avoidance"](#)), the *Protocol Layer* **Shall** either:

- Transition to the **PRL_Rx_Check_MessageID** state or
- Transition to the **PRL_Rx_Wait_for_PHY_Message** state.

6.12.2.3.4 PRL_Rx_Check_MessageID state

On entry to the **PRL_Rx_Check_MessageID** state the *Protocol Layer* **Shall** compare the **MessageID** of the received Message with its stored value if a value has previously been stored.

The *Protocol Layer* **Shall** transition to the **PRL_Rx_Wait_for_PHY_Message** state when:

- The **MessageID** of the received Message equals the stored **MessageID** value since this is a Message retry which **Shall** be **Discarded**.

The *Protocol Layer* **Shall** transition to the **PRL_Rx_Store_MessageID** state when:

- The **MessageID** of the received Message does not equal the stored **MessageID** value since this is a new Message or
- This is the first received Message and no **MessageID** value is currently stored.

6.12.2.3.5 PRL_Rx_Store_MessageID state

On entry to the **PRL_Rx_Store_MessageID** state the *Protocol Layer* **Shall** transition *Protocol Layer Message* transmission to the **PRL_Tx_Discard_Message** state, replace the stored value of **MessageID** with the value of **MessageID** in the received Message and pass the Message up to the *Policy Engine*.

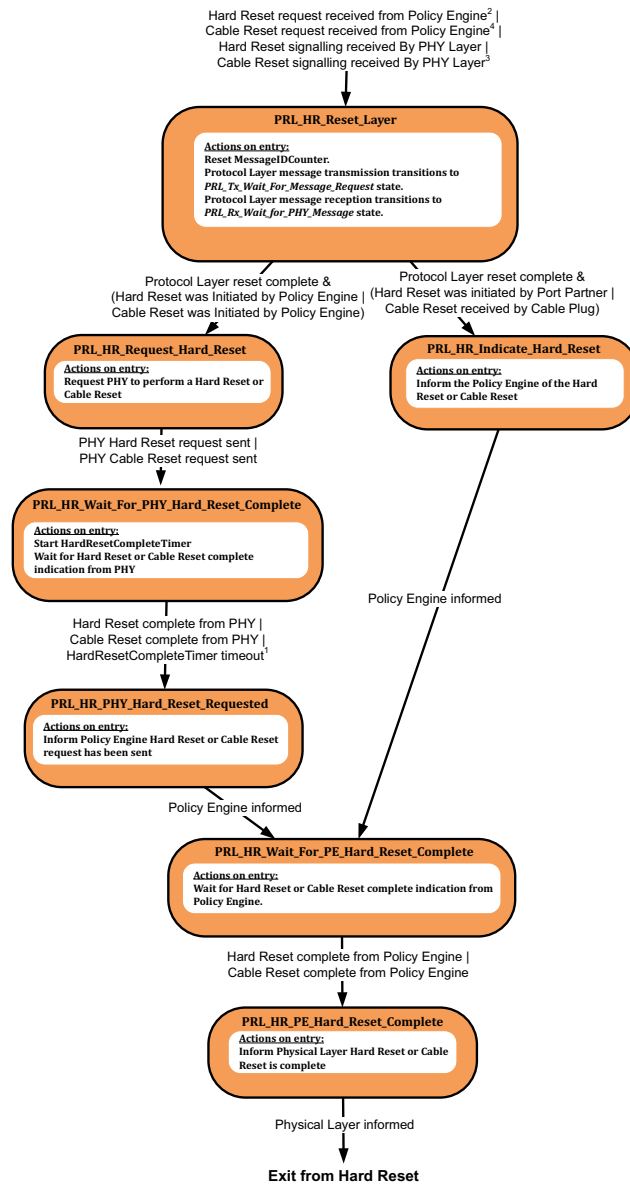
The *Protocol Layer* **Shall** transition to the **PRL_Rx_Wait_for_PHY_Message** state when:

- The Message has been passed up to the *Policy Engine*.

6.12.2.4 Hard Reset operation

[Figure 6.57, "Outline of States"](#) shows the state behavior for the *Protocol Layer* when receiving a *Hard Reset* or *Cable Reset* request from the *Policy Engine* or **Hard Reset** Signaling or **Cable Reset** Signaling from the *PHY Layer* (see also [Section 6.8.3, "Hard Reset"](#) and [Section 6.8.4, "Cable Reset"](#)).

Figure 6.67 Hard/Cable Reset



- ¹ If the *HardResetCompleteTimer* timeout occurs this means that the PHY is still waiting to send the Hard Reset due to a non-idle channel. This condition will be cleared once the PE Hard Reset is completed.
² Cable Plugs do not generate *Hard Reset* signaling but are required to monitor for *Hard Reset* signaling between the Port Partners and respond by resetting.
³ Cable Reset signaling is only recognized by a Cable Plug.
⁴ Cable Reset signaling cannot be generated by Cable Plugs.

6.12.2.4.1 PRL_HR_Reset_Layer state

The **PRL_HR_Reset_Layer** State defines the mode of operation of both the *Protocol Layer* transmission and reception state machines during a *Hard Reset* or *Cable Reset*. During *Hard Reset* no USB Power Delivery Protocol Messages are sent or received; only *Hard Reset Signaling* is present after which the communication channel is assumed to have been disabled by the *PHY Layer* until completion of the *Hard Reset*. During *Cable Reset* no USB Power Delivery Protocol Messages are sent to or received by the *Cable Plug* but other USB Power Delivery communication *May* continue.

The *Protocol Layer* **Shall** enter the **PRL_HR_Reset_Layer** state from any other state when:

- A *Hard Reset* Request is received from the *Policy Engine* or
- *Hard Reset Signaling* is received from the *PHY Layer* or

- A *Cable Reset* Request is received from the *Policy Engine* or
- *Cable Reset* Signaling is received from the *PHY Layer*.

On entry to the *PRL_HR_Reset_Layer* state the *Protocol Layer* **Shall** reset the *MessageIDCounter*. It **Shall** also reset the states of the *Protocol Layer* transmission and reception state machines to their starting points. The *Protocol Layer* transmission state machine **Shall** transition to the *PRL_Tx_Wait_for_Message_Request* state. The *Protocol Layer* reception state machine **Shall** transition to the *PRL_Rx_Wait_for_PHY_Message* state.

The *Protocol Layer* **Shall** transition to the *PRL_HR_Request_Hard_Reset* state when:

- The *Protocol Layer*'s reset is complete and
- The *Hard Reset* request has originated from the *Policy Engine* or
- The *Cable Reset* request has originated from the *Policy Engine*.

The *Protocol Layer* **Shall** transition to the *PRL_HR_Indicate_Hard_Reset* state when:

- The *Protocol Layer*'s reset is complete and
- The *Hard Reset* request has been passed up from the *PHY Layer* or
- A *Cable Reset* request has been passed up from the *PHY Layer* (*Cable Plug* only).

6.12.2.4.2 PRL_HR_Indicate_Hard_Reset state

On entry to the *PRL_HR_Indicate_Hard_Reset* state the *Protocol Layer* **Shall** indicate to the *Policy Engine* that either *Hard Reset* Signaling or *Cable Reset* Signaling has been received.

The *Protocol Layer* **Shall** transition to the *PRL_HR_Wait_for_PE_Hard_Reset_Complete* state when:

- The indication to the *Policy Engine* has been sent.

6.12.2.4.3 PRL_HR_Request_Hard_Reset state

On entry to the *PRL_HR_Request_Hard_Reset* state the *Protocol Layer* **Shall** request the *PHY Layer* to send either *Hard Reset* Signaling or *Cable Reset* Signaling.

The *Protocol Layer* **Shall** transition to the *PRL_HR_Wait_for_PHY_Hard_Reset_Complete* state when:

- The *PHY Layer* *Hard Reset* Signaling request has been sent or
- The *PHY Layer* *Cable Reset* Signaling request has been sent.

6.12.2.4.4 PRL_HR_Wait_for_PHY_Hard_Reset_Complete state

In the *PRL_HR_Wait_for_PHY_Hard_Reset_Complete* state the *Protocol Layer* **Shall** start the *HardResetCompleteTimer* and wait for the *PHY Layer* to indicate that the *Hard Reset* or *Cable Reset* has been completed.

The *Protocol Layer* **Shall** transition to the *PRL_HR_PHY_Hard_Reset_Requested* state when:

- A *Hard Reset* complete indication is received from the *PHY Layer* or
- A *Cable Reset* complete indication is received from the *PHY Layer* or
- The *HardResetCompleteTimer* times out.

6.12.2.4.5 PRL_HR_PHY_Hard_Reset_Requested state

On entry to the *PRL_HR_PHY_Hard_Reset_Requested* state the *Protocol Layer* **Shall** inform the *Policy Engine* that the *PHY Layer* has been requested to perform a *Hard Reset* or *Cable Reset*.

The *Protocol Layer* **Shall** transition to the *PRL_HR_Wait_for_PE_Hard_Reset_Complete* state when:

- The Indication to the *Policy Engine* has been sent.

6.12.2.4.6 PRL_HR_Wait_for_PE_Hard_Reset_Complete state

In the *PRL_HR_Wait_for_PE_Hard_Reset_Complete* state the *Protocol Layer* **Shall** wait for the *Policy Engine* to indicate that the *Hard Reset* or *Cable Reset* has been completed.

The *Protocol Layer* **Shall** transition to the *PRL_HR_PE_Hard_Reset_Complete* state when:

- A *Hard Reset* complete indication is received from the *Policy Engine* or
- A *Cable Reset* complete indication is received from the *Policy Engine*.

6.12.2.4.7 PRL_HR_PE_Hard_Reset_Complete

On entry to the *PRL_HR_PE_Hard_Reset_Complete* state the *Protocol Layer* **Shall** inform the *PHY Layer* that the *Hard Reset* or *Cable Reset* is complete.

The *Protocol Layer* **Shall** exit from the *Hard Reset* and return to normal operation when:

- The *PHY Layer* has been informed that the *Hard Reset* is complete so that it will re-enable the communications channel. If *Hard Reset Signaling* is still pending due to a non-*Idle* channel this **Shall** be cleared and not sent or
- The *PHY Layer* has been informed that the *Cable Reset* is complete.

6.12.3 List of Protocol Layer States

Table 6.75, "Protocol Layer States" lists the states used by the various state machines.

Table 6.75 Protocol Layer States

State Name	Section
Protocol Layer Message Transmission	
Common Protocol Layer Message Transmission	
<i>PRL_Tx_PHY_Layer_Reset</i>	Section 6.12.2.2.1.1
<i>PRL_Tx_Wait_for_Message_Request</i>	Section 6.12.2.2.1.2
<i>PRL_Tx_Layer_Reset_for_Transmit</i>	Section 6.12.2.2.1.3
<i>PRL_Tx_Construct_Message</i>	Section 6.12.2.2.1.4
<i>PRL_Tx_Wait_for_PHY_Response</i>	Section 6.12.2.2.1.5
<i>PRL_Tx_Match_MessageID</i>	Section 6.12.2.2.1.6
<i>PRL_Tx_Message_Sent</i>	Section 6.12.2.2.1.7
<i>PRL_Tx_Check_RetryCounter</i>	Section 6.12.2.2.1.8
<i>PRL_Tx_Transmission_Error</i>	Section 6.12.2.2.1.9
<i>PRL_Tx_Discard_Message</i>	Section 6.12.2.2.1.10
Source Protocol Layer Message Transmission	
<i>PRL_Tx_Src_Sink_Tx</i>	Section 6.12.2.2.2.1
<i>PRL_Tx_Src_Source_Tx</i>	Section 6.12.2.2.2.2
<i>PRL_Tx_Src_Pending</i>	Section 6.12.2.2.2.3
Sink Protocol Layer Message Transmission	
<i>PRL_Tx_Snk_Start_of_AMS</i>	Section 6.12.2.2.3.1
<i>PRL_Tx_Snk_Pending</i>	Section 6.12.2.2.3.2
Protocol Layer Message Reception	
<i>PRL_Rx_Wait_for_PHY_Message</i>	Section 6.12.2.3.1
<i>PRL_Rx_Layer_Reset_for_Receive</i>	Section 6.12.2.3.2
<i>PRL_Rx_Send_GoodCRC</i>	Section 6.12.2.3.3
<i>PRL_Rx_Check_MessageID</i>	Section 6.12.2.3.4
<i>PRL_Rx_Store_MessageID</i>	Section 6.12.2.3.5
Hard Reset Operation	
<i>PRL_HR_Reset_Layer</i>	Section 6.12.2.4.1
<i>PRL_HR_Indicate_Hard_Reset</i>	Section 6.12.2.4.2
<i>PRL_HR_Request_Hard_Reset</i>	Section 6.12.2.4.3
<i>PRL_HR_Wait_for_PHY_Hard_Reset_Complete</i>	Section 6.12.2.4.4
<i>PRL_HR_PHY_Hard_Reset_Requested</i>	Section 6.12.2.4.5
<i>PRL_HR_Wait_for_PE_Hard_Reset_Complete</i>	Section 6.12.2.4.6
<i>PRL_HR_PE_Hard_Reset_Complete</i>	Section 6.12.2.4.7

Table 6.75 Protocol Layer States (Continued)

State Name	Section
Chunking	
Chunked Rx	
<i>RCH_Wait_For_Message_From_Protocol_Layer</i>	Section 6.12.2.2.1.1
<i>RCH_Pass_Up_Message</i>	Section 6.12.2.2.1.1
<i>RCH_Processing_Extended_Message</i>	Section 6.12.2.2.1.1
<i>RCH_Requesting_Chunk</i>	Section 6.12.2.2.1.1
<i>RCH_Waiting_Chunk</i>	Section 6.12.2.2.1.1
<i>RCH_Report_Error</i>	Section 6.12.2.2.1.1
Chunked Tx	
<i>TCH_Wait_For_Message_Request_From_Policy_Engine</i>	Section 6.12.2.1.3.1
<i>TCH_Pass_Down_Message</i>	Section 6.12.2.1.3.2
<i>TCH_Wait_For_Transmission_Complete</i>	Section 6.12.2.1.3.3
<i>TCH_Message_Sent</i>	Section 6.12.2.1.3.4
<i>TCH_Prepave_To_Send_Chunked_Message</i>	Section 6.12.2.1.3.5
<i>TCH_Construct_Chunked_Message</i>	Section 6.12.2.1.3.6
<i>TCH_Sending_Chunked_Message</i>	Section 6.12.2.1.3.7
<i>TCH_Wait_Chunk_Request</i>	Section 6.12.2.1.3.8
<i>TCH_Message_Received</i>	Section 6.12.2.1.3.9
<i>TCH_Report_Error</i>	Section 6.12.2.1.3.10
Chunked Message Router	
<i>RTR_Wait_for_Message_From_Protocol_Layer</i>	Section 6.12.2.1.4.1
<i>RTR_Rx_Chunks</i>	Section 6.12.2.1.4.2
<i>RTR_Tx_Chunks</i>	Section 6.12.2.1.4.3

6.13 Message Applicability

The following tables outline the *Messages* supported by a given *Port*, depending on its capability.

When a *Message* is supported the feature and the *AMS* implied by the *Message* **Shall** also be supported. The abbreviations in [Table 6.76, "Message Applicability Abbreviations"](#) are used in this section to denote the level of support required.

Table 6.76 Message Applicability Abbreviations

Abbreviation	Meaning	Description
<i>N</i>	Normative	Shall be supported by this <i>Port/Cable Plug</i> .
<i>CN</i>	Conditional Normative	Shall supported by a given <i>Port/Cable Plug</i> based on features.
<i>R</i>	Recommended	Should be supported by this <i>Port/Cable Plug</i> .
<i>O</i>	Optional	May be supported by this <i>Port/Cable Plug</i> .
<i>NS</i>	Not Supported	Shall result in a Not Supported <i>Message</i> response by this <i>Port/Cable Plug</i> when received.
<i>I</i>	Ignore	Shall be Ignored by this <i>Port/Cable Plug</i> when received.
<i>NK</i>	NAK	This <i>Port/Cable Plug</i> Shall return Responder NAK to this <i>Command</i> when received.
<i>NA</i>	Not allowed	Shall Not be transmitted by this <i>Port/Cable Plug</i> .
<i>DR</i>	Don't Recognize	There Shall be no response at all (i.e., not even a GoodCRC <i>Message</i>) from this <i>Port/Cable Plug</i> when received.

For the case of **Conditional Normative** a note has been added to indicate the condition. "CN/" notation is used to indicate the level of support when the condition is not present.

"R/" and "O/" notation is used to indicate the response when the Recommended or **Optional** *Message* is not supported.

Note: Where NS/R/NK is indicated for Received *Messages* this **Shall** apply to the **PE_CBL_Ready**, **PE_SNK_Ready** or **PE_SRC_Ready** states only since unexpected *Messages* received during an *AMS* are *Protocol Errors* (see [Section 6.8.1, "Soft Reset and Protocol Error"](#)).

This section covers *Control Message* and *Data Message* support for *Sources*, *Sink* and *Cable Plugs*. It also covers *VDM Command* support for *DFPs*, *UFPs* and *Cable Plugs*.

6.13.1 Applicability of Control Messages

Table 6.77, "Applicability of Control Messages" details *Control Messages* that **Shall/Should/Shall Not** be transmitted and received by a *Source, Sink, Cable Plug* or *VPD*. Requirements for *Dual-Role Power Ports* and *Dual-Role Data Ports* **Shall** override any requirements for *Source-only* or *Sink-Only* Ports.

Table 6.77 Applicability of Control Messages

Message Type	Source	Sink	Dual-Role Power	Dual-Role Data	Cable Plug	VPD ⁹
Transmitted Message						
<i>Accept</i>	N	N			N	N
<i>Data_Reset</i>	CN ¹⁰ /R	CN ¹⁰ /R			NA	NA
<i>DR_Swap</i>	O	O		N	NA	NA
<i>FR_Swap</i>	NA	NA	R		NA	NA
<i>Get_Country_Codes</i>	CN ⁷ /NA	CN ⁷ /NA			NA	NA
<i>Get_PPS_Status</i>	NA	CN ⁶			NA	NA
<i>Get_Sink_Cap</i>	R	NA	N		NA	NA
<i>Get_Sink_Cap_Extended</i>	R	NA	R		NA	NA
<i>Get_Source_Cap</i>	NA	R	N		NA	NA
<i>Get_Source_Cap_Extended</i>	NA	R	R		NA	NA
<i>Get_Source_Info</i>	NA	R	R		NA	NA
<i>Get_Revision</i>	R	R			NA	NA
<i>Get_Status</i>	R	R			NA	NA
<i>GoodCRC</i>	N	N			N	N
<i>GotoMin (Deprecated)</i>	NA	NA			NA	NA
<i>Not_Supported</i>	N	N			NA	NA
<i>Ping (Deprecated)</i>	NA	NA			NA	NA
<i>PR_Swap</i>	NA	NA	N		NA	NA
<i>PS_RDY</i>	N	CN ¹ /NA	N		NA	NA
<i>Reject</i>	N	O	O	O	CN ¹⁰ /NA	NA
<i>Soft_Reset</i>	N	N			NA	NA
<i>VCONN_Swap</i>	R	R			NA	NA
<i>Wait</i>	O	NA	O	O	NA	NA

- 1) **Shall** be supported by any *Port* that can supply *VCONN*.
- 2) **Shall** be supported products that support the *Source_Capabilities_Extended* Message.
- 3) **Shall** be supported by *Sources* that support the *Alert* Message.
- 4) **Shall** be supported when the *Fast Role Swap Request* is supported.
- 5) **Shall** be supported when *VCONN Swap* is supported.
- 6) **Shall** be supported when *SPR PPS Mode* is supported.
- 7) **Shall** be supported when required by a country authority.
- 8) **Shall** be supported by *Active Cables*.
- 9) *VPD* includes *CT-VPDs* when not *Connected* to a *Charger*. PD communication with a *CT-VPD* **Shall** only take place when not *Connected* to a *Charger*.
- 10) **Shall** be supported by products that support *[USB4]*.
- 11) **Shall** be supported by all *Sources* except single *Port SPR Chargers* with *Invariant PDOs*.

Table 6.77 Applicability of Control Messages (Continued)

Message Type	Source	Sink	Dual-Role Power	Dual-Role Data	Cable Plug	VPD ⁹
Received Message						
<i>Accept</i>	N	N	N	N	I	I
<i>Data_Reset</i>	CN ¹⁰ /R	CN ¹⁰ /R			I	I
<i>DR_Swap</i>	O/NS	O/NS		N	I	I
<i>FR_Swap</i>	NS	NS	CN ⁴ /NS		I	I
<i>Get_Country_Codes</i>	CN ⁷ /NS	CN ⁷ /NS			I	I
<i>Get_PPS_Status</i>	CN ⁶ /NS	NS			I	I
<i>Get_Sink_Cap</i>	NS	N	N		I	I
<i>Get_Sink_Cap_Extended</i>	NS	N	N		I	I
<i>Get_Source_Cap</i>	N	NS	N		I	I
<i>Get_Source_Cap_Extended</i>	CN ² /NS	NS	CN ² /NS		I	I
<i>Get_Source_Info</i>	CN ¹¹	NS	N		I	I
<i>Get_Revision</i>	N	N			O/I	O/I
<i>Get_Status</i>	CN ³ /NS	CN ³ /NS	CN ³ /NS		CN ⁸ /I	I
<i>GoodCRC</i>	N	N			N	N
<i>GotoMin (Deprecated)</i>	NS	NS			I	I
<i>Not_Supported</i>	N	N			CN ⁸ /I	I
<i>Ping (Deprecated)</i>	NS	NS/I			I	I
<i>PR_Swap</i>	NS	NS	N		I	I
<i>PS_RDY</i>	CN ¹ /NS	N	N		I	I
<i>Reject</i>	CN ⁵ /NS	N	N	N	I	I
<i>Soft_Reset</i>	N	N			N	N
<i>VCONN_Swap</i>	CN ¹ /NS	CN ¹ /NS			I	I
<i>Wait</i>	CN ⁵ /NS	N	N	N	I	I

- 1) **Shall** be supported by any *Port* that can supply *VCONN*.
- 2) **Shall** be supported products that support the *Source_Capabilities_Extended* Message.
- 3) **Shall** be supported by *Sources* that support the *Alert* Message.
- 4) **Shall** be supported when the *Fast Role Swap Request* is supported.
- 5) **Shall** be supported when *VCONN Swap* is supported.
- 6) **Shall** be supported when *SPR PPS Mode* is supported.
- 7) **Shall** be supported when required by a country authority.
- 8) **Shall** be supported by *Active Cables*.
- 9) *VPD* includes *CT-VPDs* when not *Connected* to a *Charger*. PD communication with a *CT-VPD* **Shall** only take place when not *Connected* to a *Charger*.
- 10) **Shall** be supported by products that support *[USB4]*.
- 11) **Shall** be supported by all *Sources* except single *Port SPR Chargers* with *Invariant PDOs*.

6.13.2 Applicability of Data Messages

[Table 6.78, "Applicability of Data Messages"](#) details *Data Messages* (except for *VDM Commands*) that **Shall/Should/ Shall Not** be transmitted and received by a *Source*, *Sink*, *Cable Plug* or *VPD*. Requirements for *Dual-Role Power Ports* **Shall** override any requirements for *Source-only* or *Sink-Only* Ports.

Table 6.78 Applicability of Data Messages

Message Type	Source	Sink	Dual-Role Power	Cable Plug SOP'	Cable Plug SOP''	VPD ⁶
Transmitted Message						
<i>Source_Capabilities</i>	N	NA	N	NA	NA	NA
<i>Request</i>	NA	N		NA	NA	NA
<i>Get_Country_Info</i>	CN ⁵ /O	CN ⁵ /O		NA	NA	NA
<i>BIST</i>	N ¹	N ¹		NA	NA	NA
<i>Sink_Capabilities</i>	NA	N	N	NA	NA	NA
<i>Battery_Status</i>	CN ²	CN ²		NA	NA	NA
<i>Alert</i>	CN ¹¹ /R	CN ¹¹ /R		NA	NA	NA
<i>Enter_USB</i>	CN ⁷ /O	CN ⁷ /O		NA	NA	NA
<i>EPR_Request</i>	NA	CN ⁹		NA	NA	NA
<i>EPR_Mode</i>	CN ⁹	CN ⁹		NA	NA	NA
<i>Source_Info</i>	CN ¹⁰	NA	N	NA	NA	NA
<i>Revision</i>	N	N		CN ¹² /O/I	NA	O
Received Message						
<i>Source_Capabilities</i>	NS	N	N	I	I	I
<i>Request</i>	N	NS		I	I	I
<i>Get_Country_Info</i>	CN ⁵ /NS	CN ⁵ /NS		I	I	I
<i>BIST</i>	N ¹	N ¹		N ¹	N ¹	N ¹
<i>Sink_Capabilities</i>	CN ⁴	NS	CN ⁴	I	I	I
<i>Battery_Status</i>	CN ³ /NS	CN ³ /NS		I	I	I
<i>Alert</i>	R/NS	R/NS		I	I	I
<i>Enter_USB</i>	CN ⁷ /O	CN ⁷ /O		CN ⁸ /I	CN ⁸ /I	I

- 1) For details of which *BIST Modes* and *BIST Messages* **Shall** be supported see [Section 5.9](#) and [Section 6.4.3](#).
- 2) **Shall** be supported by products that contain batteries.
- 3) **Shall** be supported by products that support the *Get_Battery_Status* Message.
- 4) **Shall** be supported by products that support the *Get_Sink_Cap* Message.
- 5) **Shall** be supported when required by a country authority.
- 6) *VPD* includes *CT-VPDs* when not *Connected* to a *Charger*. PD communication with a *CT-VPD* **Shall** only take place when not *Connected* to a *Charger*.
- 7) **Shall** be supported by products that support [\[USB4\]](#).
- 8) **Shall** be supported by *Active Cables* that support [\[USB4\]](#).
- 9) **Shall** be supported by products that support *Source* operation in *EPR Mode*.
- 10) **Shall** be supported by all *Source Ports* except singlePort *SPR Chargers* with *Invariant PDOs*.
- 11) **Shall** be supported when *SPR PPS Mode* is supported.
- 12) **Shall** be supported by *Active Cables*.

Table 6.78 Applicability of Data Messages (Continued)

Message Type	Source	Sink	Dual-Role Power	Cable Plug SOP'	Cable Plug SOP''	VPD ⁶
<i>EPR_Request</i>	CN ⁹	NA		I	I	I
<i>EPR_Mode</i>	CN ⁹	CN ⁹		I	I	I
<i>Source_Info</i>	NA	N	N	I	I	I
<i>Revision</i>	N	N		I	I	I
1) For details of which <i>BIST Modes</i> and <i>BIST</i> Messages Shall be supported see Section 5.9 and Section 6.4.3 . 2) Shall be supported by products that contain batteries. 3) Shall be supported by products that support the <i>Get_Battery_Status</i> Message. 4) Shall be supported by products that support the <i>Get_Sink_Cap</i> Message. 5) Shall be supported when required by a country authority. 6) VPD includes <i>CT-VPDs</i> when not <i>Connected</i> to a <i>Charger</i> . PD communication with a <i>CT-VPD</i> Shall only take place when not <i>Connected</i> to a <i>Charger</i> . 7) Shall be supported by products that support [USB4] . 8) Shall be supported by <i>Active Cables</i> that support [USB4] . 9) Shall be supported by products that support <i>Source</i> operation in <i>EPR Mode</i> . 10) Shall be supported by all <i>Source Ports</i> except singlePort <i>SPR Chargers</i> with <i>Invariant PDOs</i> . 11) Shall be supported when <i>SPR PPS Mode</i> is supported. 12) Shall be supported by <i>Active Cables</i> .						

6.13.3 Applicability of Extended Messages

Table 6.79, "Applicability of Extended Messages" details *Extended Messages* (except for *VDEM Commands*) that **Shall/Should/ Shall Not** be transmitted and received by a *Source*, *Sink*, *Cable Plug* or *VPD*. Requirements for *Dual-Role Power Ports* **Shall** override any requirements for *Source-only* or *Sink-Only* Ports.

Table 6.79 Applicability of Extended Messages

Message Type	Source	Sink	Dual-Role Power	Cable Plug SOP'	Cable Plug SOP''	VPD ¹³
Transmitted Message						
<i>Battery_Capabilities</i>	CN ¹ /NA	CN ¹ /NA		NA	NA	NA
<i>Country_Codes</i>	CN ¹⁰ /NA	CN ¹⁰ /NA		NA	NA	NA
<i>Country_Info</i>	CN ¹⁰ /NA	CN ¹⁰ /NA		NA	NA	NA
<i>EPR_Source_Capabilities</i>	CN ¹⁴ /NA	NA	CN ¹⁴ /NA	NA	NA	NA
<i>EPR_Sink_Capabilities</i>	NA	CN ¹⁴ /NA	CN ¹⁴ /NA	NA	NA	NA
<i>Extended_Control</i>	See Section 6.13.4 for details					
<i>Firmware_Update_Request</i>	CN ⁷ /NA	CN ⁷ /NA		NA	NA	NA
<i>Firmware_Update_Response</i>	CN ⁷ /NA	CN ⁷ /NA		CN ⁷ /NA	O	NA
<i>Get_Battery_Cap</i>	R	R		NA	NA	NA
<i>Get_Battery_Status</i>	R	R		NA	NA	NA
<i>Get_Manufacturer_Info</i>	R	R		NA	NA	NA
<i>Manufacturer_Info</i>	R	R		R	NA	NA
<i>PPS_Status</i>	CN ⁸ /NA	NA		NA	NA	NA
<i>Security_Request</i>	CN ⁶ /NA	CN ⁶ /NA		NA	NA	NA
<i>Security_Response</i>	CN ⁶ /NA	CN ⁶ /NA		CN ⁶ /NA	NA	NA
<i>Sink_Capabilities_Extended</i>	NA	N	N	NA	NA	NA
<i>Source_Capabilities_Extended</i>	R	NA	R	NA	NA	NA
<ol style="list-style-type: none"> 1) Shall be supported by products that contain batteries. 2) Shall be supported by products that can transmit the <i>Get_Source_Cap_Extended</i> Message. 3) Shall be supported by products that can transmit the <i>Get_Status</i> Message. 4) Shall be supported by products that can transmit the <i>Get_Battery_Cap</i> Message. 5) Shall be supported by products that can transmit the <i>Get_Manufacturer_Info</i> Message. 6) Shall be supported by products that support USB security communication as defined in [USBTypeCAuthentication 1.0]. 7) Shall be supported by products that support USB firmware update communication as defined in [USBPDFirmwareUpdate 1.0]. 8) Shall be supported when PPS is supported. 9) Shall be supported by products that can transmit the <i>Get_PPS_Status</i> Message. 10) Shall be supported when required by a country authority. 11) Shall be supported by products that can transmit the <i>Get_Sink_Cap_Extended</i> Message. 12) Shall be supported by <i>Active Cables</i>. 13) <i>VPD</i> includes <i>CT-VPDs</i> when not <i>Connected</i> to a <i>Charger</i>. PD communication with a <i>CT-VPD</i> Shall only take place when not <i>Connected</i> to a <i>Charger</i>. 14) Shall be supported by products that support operation in <i>EPR Mode</i>. 15) Shall be supported by <i>Sources</i> that support the <i>Alert</i> Message. 						

Table 6.79 Applicability of Extended Messages (Continued)

Message Type	Source	Sink	Dual-Role Power	Cable Plug SOP'	Cable Plug SOP''	VPD ¹³
<i>Status</i>	CN ¹⁵ /R	CN ¹⁵ /R	CN ¹⁵ /R	CN ¹² /NA	CN ¹² /NA	NA
<i>Vendor_Defined_Extended</i>	O	O		O	O	O
Received Message						
<i>Battery_Capabilities</i>	CN ⁴ /NS	CN ⁴ /NS		I	I	I
<i>Country_Codes</i>	CN ¹⁰ /NS	CN ¹⁰ /NS		I	I	I
<i>Country_Info</i>	CN ¹⁰ /NS	CN ¹⁰ /NS		I	I	I
<i>EPR_Source_Capabilities</i>	NS	CN ¹⁴ /NS	CN ¹⁴ /NS	I	I	I
<i>EPR_Sink_Capabilities</i>	CN ¹⁴ /NS	NS	CN ¹⁴ /NS	I	I	I
<i>Extended_Control</i>	See Section 6.13.4 for details					
<i>Firmware_Update_Request</i>	CN ⁷ /NS	CN ⁷ /NS		CN ⁷ /I	O	I
<i>Firmware_Update_Response</i>	CN ⁷ /NS	CN ⁷ /NS		I	I	I
<i>Get_Battery_Cap</i>	CN ¹ /NS	CN ¹ /NS		I	I	I
<i>Get_Battery_Status</i>	CN ¹ /NS	CN ¹ /NS		I	I	I
<i>Get_Manufacturer_Info</i>	R/NS	R/NS		R/I	I	I
<i>Manufacturer_Info</i>	CN ⁵ /NS	CN ⁵ /NS		I	I	I
<i>PPS_Status</i>	NS	CN ⁹ /NS		I	I	I
<i>Security_Request</i>	CN ⁶ /NS	CN ⁶ /NS		CN ⁶ /I	I	I
<i>Security_Response</i>	CN ⁶ /NS	CN ⁶ /NS		I	I	I
<i>Sink_Capabilities_Extended</i>	CN ¹¹ /NS	NS	CN ¹¹ /NS	I	I	I
<i>Source_Capabilities_Extended</i>	NS	CN ² /NS	CN ² /NS	I	I	I
<ol style="list-style-type: none"> 1) Shall be supported by products that contain batteries. 2) Shall be supported by products that can transmit the <i>Get_Source_Cap_Extended</i> Message. 3) Shall be supported by products that can transmit the <i>Get_Status</i> Message. 4) Shall be supported by products that can transmit the <i>Get_Battery_Cap</i> Message. 5) Shall be supported by products that can transmit the <i>Get_Manufacturer_Info</i> Message. 6) Shall be supported by products that support USB security communication as defined in [USBTypeCAuthentication 1.0]. 7) Shall be supported by products that support USB firmware update communication as defined in [USBPDFirmwareUpdate 1.0]. 8) Shall be supported when PPS is supported. 9) Shall be supported by products that can transmit the <i>Get_PPS_Status</i> Message. 10) Shall be supported when required by a country authority. 11) Shall be supported by products that can transmit the <i>Get_Sink_Cap_Extended</i> Message. 12) Shall be supported by <i>Active Cables</i>. 13) VPD includes CT-VPDs when not <i>Connected</i> to a <i>Charger</i>. PD communication with a CT-VPD Shall only take place when not <i>Connected</i> to a <i>Charger</i>. 14) Shall be supported by products that support operation in <i>EPR Mode</i>. 15) Shall be supported by <i>Sources</i> that support the <i>Alert</i> Message. 						

Table 6.79 Applicability of Extended Messages (Continued)

Message Type	Source	Sink	Dual-Role Power	Cable Plug SOP'	Cable Plug SOP''	VPD ¹³
<i>Status</i>	CN ³³ /NS	CN ³ /NS		I	I	I
<i>Vendor_Defined_Extended</i>	O/NS	O/NS		O/I	O/I	O/I
1) Shall be supported by products that contain batteries. 2) Shall be supported by products that can transmit the <i>Get_Source_Cap_Extended</i> Message. 3) Shall be supported by products that can transmit the <i>Get_Status</i> Message. 4) Shall be supported by products that can transmit the <i>Get_Battery_Cap</i> Message. 5) Shall be supported by products that can transmit the <i>Get_Manufacturer_Info</i> Message. 6) Shall be supported by products that support USB security communication as defined in <i>[USBTypeCAuthentication 1.0]</i> . 7) Shall be supported by products that support USB firmware update communication as defined in <i>[USBPDFirmwareUpdate 1.0]</i> . 8) Shall be supported when PPS is supported. 9) Shall be supported by products that can transmit the <i>Get_PPS_Status</i> Message. 10) Shall be supported when required by a country authority. 11) Shall be supported by products that can transmit the <i>Get_Sink_Cap_Extended</i> Message. 12) Shall be supported by <i>Active Cables</i> . 13) VPD includes <i>CT-VPDs</i> when not <i>Connected</i> to a <i>Charger</i> . PD communication with a <i>CT-VPD</i> Shall only take place when not <i>Connected</i> to a <i>Charger</i> . 14) Shall be supported by products that support operation in <i>EPR Mode</i> . 15) Shall be supported by <i>Sources</i> that support the <i>Alert</i> Message.						

6.13.4 Applicability of Extended Control Messages

[Table 6.80, "Applicability of Extended Control Messages"](#) details *Extended Control Messages* that **Shall/Should/****Shall Not** be transmitted and received by a *Source*, *Sink*, *Cable Plug* or *VPD*. Requirements for *Dual-Role Power Ports* and *Dual-Role Data Ports* **Shall** override any requirements for *Source-only* or *Sink-Only* Ports.

Table 6.80 Applicability of Extended Control Messages

Message Type	Source	Sink	Dual-Role Power	Dual-Role Data	Cable Plug	VPD ²
Transmitted Message						
<i>EPR_Get_Source_Cap</i>	NA	CN ¹	CN ¹		NA	NA
<i>EPR_Get_Sink_Cap</i>	CN ¹	NA	CN ¹		NA	NA
<i>EPR_KeepAlive</i>	NA	CN ¹			NA	NA
<i>EPR_KeepAlive_Ack</i>	CN ¹	NA			NA	NA
Received Message						
<i>EPR_Get_Source_Cap</i>	CN ¹	NS	CN ¹		I	I
<i>EPR_Get_Sink_Cap</i>	NS	CN ¹	CN ¹		I	I
<i>EPR_KeepAlive</i>	CN ¹	NS			I	I
<i>EPR_KeepAlive_Ack</i>	NS	CN ¹			I	I
1) Shall be supported by products that support <i>EPR Mode</i> . 2) <i>VPD</i> includes <i>CT-VPDs</i> when not <i>Connected</i> to a <i>Charger</i> . PD communication with a <i>CT-VPD</i> Shall only take place when not <i>Connected</i> to a <i>Charger</i> .						

6.13.5 Applicability of Structured VDM Commands

[Table 6.81, "Applicability of Structured VDM Commands"](#) details *Structured VDM Commands* that **Shall/Should/ Shall Not** be transmitted and received by a *DFP*, *UFP*, *Cable Plug* or *VPD*. If *Structured VDMs* are not supported, the *DFP* or *UFP* receiving a *VDM Command* **Shall** send a **Not_Supported** Message in response.

Table 6.81 Applicability of Structured VDM Commands

Command Type	DFP	UFP	Cable Plug SOP'	Cable Plug SOP''	VPD ⁴
Transmitted Command Request					
<i>Discover Identity</i>	CN ^{1,6} /R	R ²	NA	NA	NA
<i>Discover SVIDs</i>	CN ¹ /O	O	NA	NA	NA
<i>Discover Modes</i>	CN ¹ /O	O	NA	NA	NA
<i>Enter Mode</i>	CN ¹ /NA	NA	NA	NA	NA
<i>Exit Mode</i>	CN ¹ /NA	NA	NA	NA	NA
<i>Attention</i>	O	O	NA	NA	NA
Received Command Request/Transmitted Command Response					
<i>Discover Identity</i>	CN ^{5,6} /R/ NK ³	CN ^{1,6} /R/ NK ³	N	I	N
<i>Discover SVIDs</i>	O/NK ³	CN ¹ /NK ³	CN ¹ /NK	I	NK
<i>Discover Modes</i>	O/NK ³	CN ¹ /NK ³	CN ¹ /NK	I	NK
<i>Enter Mode</i>	NK ³	CN ¹ /NK ³	CN ¹ /NK	O	NK
<i>Exit Mode</i>	NK ³	CN ¹ /NK ³	CN ¹ /NK	O	NK
<i>Attention</i>	O/I ³	O/I ³	I	I	I
<ol style="list-style-type: none"> 1) Shall be supported when <i>Modal Operation</i> is supported. 2) May be transmitted by a <i>UFP/Source</i> during discovery (see Section 6.4.4.3.1, "Discover Identity" and Section 8.3.3.25.3, "Source Startup Structured VDM Discover Identity of a Cable Plug State Diagram"). 3) If <i>Structured VDMs</i> are not supported, the <i>DFP</i> or <i>UFP</i> receiving a <i>VDM Command</i> Shall send a Not_Supported Message in response. 4) <i>VPD</i> includes <i>CT-VPDs</i> when not <i>Connected</i> to a <i>Charger</i>. PD communication with a <i>CT-VPD</i> Shall only take place when not <i>Connected</i> to a <i>Charger</i>. 5) Shall be supported by products with more than one <i>DFP</i>. 6) Shall be supported by products that support [USB4]. 					

6.13.6 Applicability of Reset Signaling

[Table 6.82, "Applicability of Reset Signaling"](#) details the Reset that **Shall/Should/ Shall Not** be transmitted and received by a DFP/UFP or Cable Plug.

Table 6.82 Applicability of Reset Signaling

Reset Type	DFP	UFP	Cable Plug SOP'	Cable Plug SOP''	VPD ²
Transmitted Message/Signaling					
<i>Soft_Reset</i>	N	N	NA	NA	NA
<i>Hard_Reset</i>	N	N	NA	NA	NA
<i>Cable_Reset</i>	CN ¹	NA	NA	NA	NA
Received Message/Signaling					
<i>Soft_Reset</i>	N	N	N	N	N
<i>Hard_Reset</i>	N	N	N	N	N
<i>Cable_Reset</i>	DR	DR	N	N	N
1)	Shall be supported when transmission of <i>SOP' Packets</i> are supported, and the <i>Port</i> can supply <i>VCONN</i> .				
2)	<i>VPD</i> includes <i>CT-VPDs</i> when not <i>Connected</i> to a <i>Charger</i> . PD communication with a <i>CT-VPD Shall</i> only take place when not <i>Connected</i> to a <i>Charger</i> .				

6.13.7 Applicability of Fast Role Swap Request

[Table 6.83, "Applicability of Fast Role Swap Request"](#) details the *Fast Role Swap Request* that **Shall/Should/ Shall Not** be transmitted and received by a *Source* or *Sink*.

Table 6.83 Applicability of Fast Role Swap Request

Command Type	Source	Sink	Dual-Role Power
Transmitted Message/Signaling			
<i>Fast Role Swap</i>	NA	NA	R
Received Message/Signaling			
<i>Fast Role Swap</i>	NA	NA	R

6.14 Value Parameters

[Table 6.84, "Value Parameters"](#) contains value parameters used in this section.

Table 6.84 Value Parameters

Parameter	Description	Value	Unit	Reference
<i>MaxExtendedMsgLen</i>	Maximum length of an <i>Extended Message</i> as expressed in the <i>Data Size</i> field.	260	Byte	Section 6.2.1.2
<i>MaxExtendedMsgChunkLen</i>	Maximum length of an <i>Extended Message Chunk</i> .	26	Byte	Section 6.2.1.2
<i>MaxExtendedMsgLegacyLen</i>	Maximum length of an <i>Extended Message</i> that can be sent without <i>Chunking</i> .	26	Byte	Section 6.2.1.2

7 Power Supply

7.1 Source Requirements

7.1.1 Behavioral Aspects

A *PDUSB Source* exhibits the following behaviors:

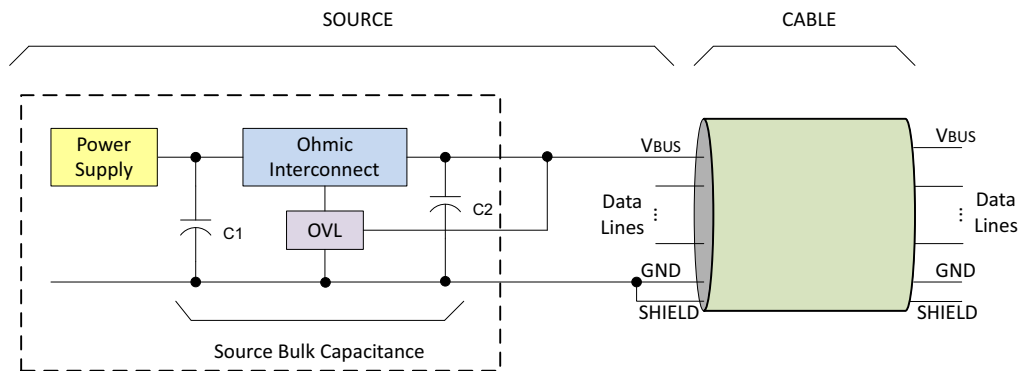
- **Shall** supply [\[USB Type-C 2.4\]](#) USB Type-C[®] current to *VBUS* while in a *Default Contract* or *Implicit Contract*.
- **Shall** follow the requirements as specified in [Section 7.1.5, "Response to Hard Resets"](#) when **Hard Reset Signaling** is received.
- **Shall** control *VBUS* voltage transitions as bound by undershoot, overshoot and transition time requirements.

7.1.2 Source Bulk Capacitance

The *Source* bulk capacitance **Shall Not** be placed between the transceiver isolation impedance and the USB receptacle. The *Source* bulk capacitance consists of C1 and C2 as shown in [Figure 7.1, "Placement of Source Bulk Capacitance"](#). The Ohmic Interconnect might consist of PCB traces for power distribution or power switching devices. The Ohmic Interconnect might also be part of the circuit implemented by the *Source* to limit its *VBUS* Output Voltage Limit (OVL) as described in [Section 7.1.7.5, "Output Voltage Limit"](#). Though a *Source* **Shall** limit its output voltage, a *Sink* **Shall** implement *Sink OVP* as described in [Section 7.2.9.2, "Input Over Voltage Protection"](#) to protect against excessive *VBUS* input voltage. The capacitance might be a single capacitor, a capacitor bank or distributed capacitance. If the power supply is shared across multiple ports, the bulk capacitance is defined as ***cSrcBulkShared***. If the power supply is dedicated to a single *Port*, the minimum bulk capacitance is defined as ***cSrcBulk***.

The *Source* bulk capacitance is allowed to change for a newly *Negotiated* power level. The capacitance change **Shall** occur before the *Source* is ready to operate at the new power level. During a *Power Role Swap*, the *Initial Source* **Shall** transition to *Swap Standby* before operating as the *New Sink*. Any change in bulk capacitance required to complete the *Power Role Swap* **Shall** occur during *Swap Standby*.

Figure 7.1 Placement of Source Bulk Capacitance



7.1.3 Types of Sources

Consistent with the *Power Data Objects* discussed in [Section 6.4.1, "Capabilities Message"](#), the power supply types that are available as *Sources* in a USB Power Delivery System are:

- The *Fixed Supply PDO* exposes well-regulated fixed voltage power supplies. *Sources* **Shall** support at least one *Fixed Supply* capable of supplying **vSafe5V**. The output voltage of a *Fixed Supply* **Shall** remain

within the range defined by the relative tolerance **vSrcNew** and the absolute band **vSrcValid** as listed in [Table 7.23, "Source Electrical Parameters"](#) and described in [Section 7.1.8, "Output Voltage Tolerance and Range"](#).

- The *Variable Supply* (non-Battery) *PDO* exposes less well-regulated *Sources*. The output voltage of a *Variable Supply* (non-Battery) **shall** remain within the absolute maximum output voltage and the absolute minimum output voltage exposed in the *Variable Supply PDO*.
- The *Battery Supply PDO* exposes Batteries than can be connected directly as a *Source* to *VBUS*. The output voltage of a *Battery Supply* **shall** remain within the absolute maximum output voltage and the absolute minimum output exposed in the *Battery Supply PDO*.
- The Programmable Power Supply (PPS) *Augmented Power Data Object (APDO)* exposes a *Source* with an output voltage that can be adjusted programmatically over a defined range. The output voltage of the Programmable Power Supply **shall** remain within a range defined by the relative tolerance **vPpsNew** and the absolute band **vPpsValid**.
- The *Adjustable Voltage Supply (AVS) Augmented Power Data Object (APDO)* exposes a *Source* with an output voltage that can be adjusted programmatically over a defined range. The output voltage of the *AVS* **shall** remain within a range defined by the relative tolerance **vAvsNew** and the absolute band **vAvsValid**.

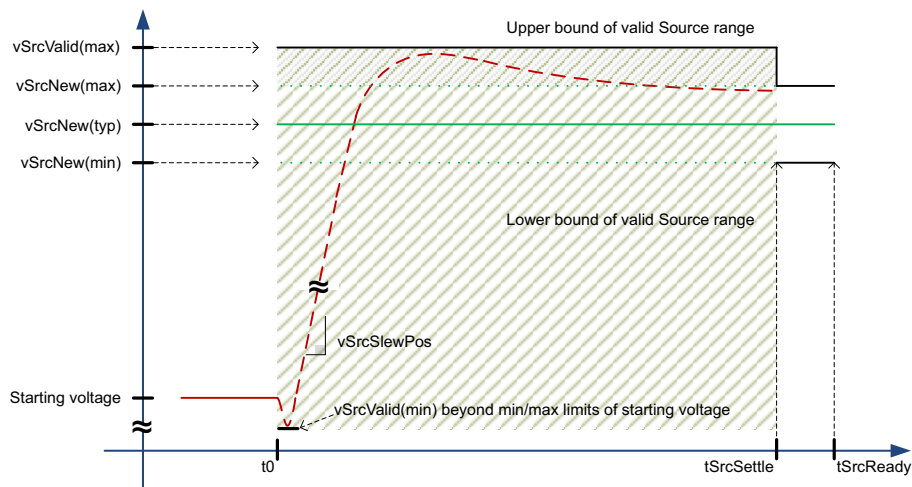
7.1.4 Source Transitions

7.1.4.1 Fixed Supply

7.1.4.1.1 Fixed Supply Positive Voltage Transitions

The *Source* **shall** transition *VBUS* from the starting voltage to the higher new voltage in a controlled manner. The *Negotiated* new voltage (e.g., 5V, 9V, 15V, ...) defines the nominal value for **vSrcNew**. During the positive transition the *Source* **should** be able to supply the *Sink Standby* current and the transient current to charge the total bulk capacitance on *VBUS*. The slew rate of the positive transition **shall not** exceed **vSrcSlewPos**. The transitioning *Source* output voltage **shall** settle within **vSrcNew** by **tSrcSettle**. The *Source* **shall** be able to supply the *Negotiated* power level at the new voltage by **tSrcReady**. The positive voltage transition **shall** remain above **vSrcValid** min of the previous *Explicit Contract* and below **vSrcValid** max of the new *Explicit Contract* ([Figure 7.2, "Transition Envelope for Positive Voltage Transitions"](#)). The voltage **shall** settle to **vSrcNew** within **tSrcSettle**. The starting time, t_0 , in [Figure 7.2, "Transition Envelope for Positive Voltage Transitions"](#) starts **tSrcTransition** after the last bit of the *EOP* of the *GoodCRC* Message has been received by the *Source*.

Figure 7.2 Transition Envelope for Positive Voltage Transitions



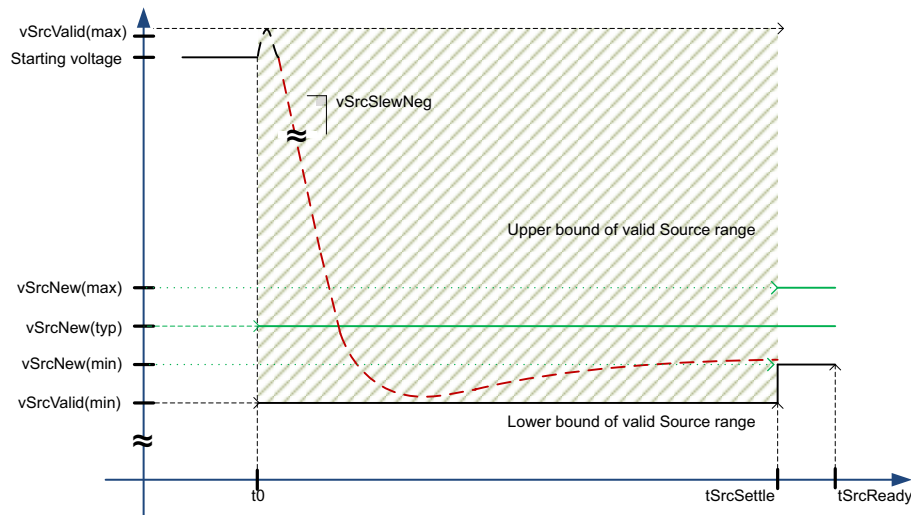
At the start of the positive voltage transition the *VBUS* voltage level **shall not** droop **vSrcValid** min below either **vSrcNew** (i.e., if the starting *VBUS* voltage level is not **vSafe5V**) or **vSafe5V** as applicable.

[Section 7.1.14, "Non-application of \$V_{BUS}\$ Slew Rate Limits"](#) lists transitions that are exempt from the $v_{SrcSlewPos}$ limit.

7.1.4.1.2 Fixed Supply Negative Voltage Transitions

Negative voltage transitions are defined as shown in [Figure 7.3, "Transition Envelope for Negative Voltage Transitions"](#) and are specified in a similar manner to positive voltage transitions. [Figure 7.3, "Transition Envelope for Negative Voltage Transitions"](#) does not apply to v_{Safe0V} transitions. The slew rate of the negative transition **Shall Not** exceed $v_{SrcSlewNeg}$. The negative voltage transition **Shall** remain below $v_{SrcValid}$ max of the previous *Explicit Contract* and above $v_{SrcValid}$ min of the new *Explicit Contract*, as shown in [Figure 7.3, "Transition Envelope for Negative Voltage Transitions"](#). The transitioning *Source* output voltage **Shall** settle to v_{SrcNew} within $t_{SrcSettle}$. The starting time, t_0 , in [Figure 7.3, "Transition Envelope for Negative Voltage Transitions"](#) starts $t_{SrcTransition}$ after the last bit of the *EOP* of the *GoodCRC* Message has been received by the *Source*.

Figure 7.3 Transition Envelope for Negative Voltage Transitions



If the newly *Negotiated* voltage is v_{Safe5V} , then the $v_{SrcValid}$ limits **Shall** determine the transition window and the transitioning *Source* **Shall** settle within the v_{Safe5V} limits by $t_{SrcSettle}$.

[Section 7.1.14, "Non-application of \$V_{BUS}\$ Slew Rate Limits"](#) lists transitions that are exempt from the $v_{SrcSlewNeg}$ limit.

7.1.4.2 SPR Programmable Power Supply (PPS)

7.1.4.2.1 SPR Programmable Power Supply Voltage Transitions

The Programmable Power Supply (PPS) **Shall** transition V_{BUS} over the defined voltage range in a controlled manner. The Output Voltage value in the Programmable *RDO* defines the nominal value of the PPS output voltage after completing a voltage change and **Shall** settle within the limits defined by v_{PpsNew} by $t_{PpsSrcTransSmall}$ for steps smaller than or equal to $v_{PpsSmallStep}$, or else, within the limits defined by v_{PpsNew} by $t_{PpsSrcTransLarge}$, but only in case the Programmable Power Supply is not in *CL* mode. Any overshoot beyond v_{PpsNew} **Shall Not** exceed $v_{PpsValid}$ at any time. Any undershoot beyond v_{PpsNew} **Shall Not** exceed $v_{PpsValid}$ for currents not resulting in *CL* mode. The PPS output voltage **May** change in a step-wise or linear manner and the slew rate of either type of change **Shall Not** exceed $v_{PpsSlewPos}$ for voltage increases or $v_{PpsSlewNeg}$ for voltage decreases. The nominal requested voltage of all linear voltage changes **Shall** equate to an integer number of *LSB* changes. An *LSB* change of the PPS output voltage is defined as $v_{PpsStep}$. A PPS **Shall** be able to supply the *Negotiated* current level as it changes its output voltage to the requested level. All PPS voltage increases **Shall**

result in a voltage that is greater than or equal to the previous PPS output voltage. Likewise, all PPS voltage decreases **shall** result in a voltage that is less than or equal to the previous PPS output voltage.

Since a *Sink* can draw current up to the *Negotiated APDO* current level in case of a voltage step, the voltage might not increase to the requested level due to the power supply operating in *CL* mode. Likewise, since a *Sink* can have a *Battery* connected to *VBUS*, the voltage might not decrease to the requested level due to the *Battery* voltage being higher than the output voltage set point the *Source* is transitioning to. Were the *Source* to rely on checking the voltage on *VBUS*, in either case, to determine when its power supply is ready a **PS_RDY** Message would never be sent.

When the PPS voltage steps up or down, a **PS_RDY** Message **shall** be sent within:

- **tPpsSrcTransLarge** after the last bit of the **GoodCRC** Message following the **Accept** Message for steps larger than **vPpsSmallStep**.
- **tPpsSrcTransSmall** after the last bit of the **GoodCRC** Message following the **Accept** Message for steps less than or equal to **vPpsSmallStep** provided that either the voltage on *VBUS* has reached **vPpsNew** or the power supply is in *CL* mode.

When **vPpsNew** is lower than the *Battery* voltage, or the *Source*'s primary power is cut off the *Sink* **shall** immediately disconnect its *Battery* from *VBUS*. In these situations, the output current could reverse polarity and the *Sink* is not allowed to source current (see [Section 7.2.1, "Behavioral Aspects"](#) and [Section 7.2.9, "Robust Sink Operation"](#)).

[Figure 7.4, "PPS Positive Voltage Transitions"](#) and [Figure 7.5, "PPS Negative Voltage Transitions"](#) below show the output voltage behavior of a Programmable Power Supply in response to positive and negative voltage change requests. The parameters **vPpsMinVoltage** and **vPpsMaxVoltage** define the lower and upper limits of the PPS range respectively (see [Table 10.11, "SPR Programmable Power Supply Voltage Ranges"](#) for required ranges). **vPpsMinVoltage** corresponds to the **Minimum Voltage** field in the PPS APDO and **vPpsMaxVoltage** corresponds to **Maximum Voltage** field in the PPS APDO. If the *Sink* negotiates for a new PPS APDO, then the transition between the two PPS APDOs **shall** occur as described in [Section 7.3.1, "Transitions caused by a Request Message"](#).

Figure 7.4 PPS Positive Voltage Transitions

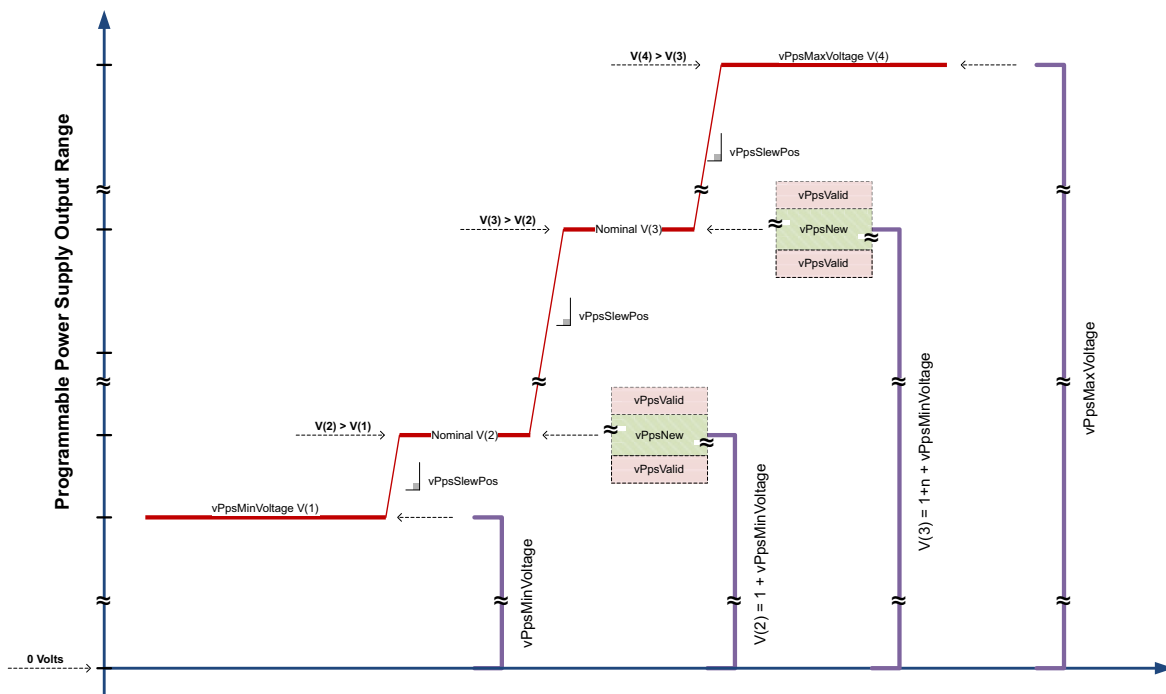
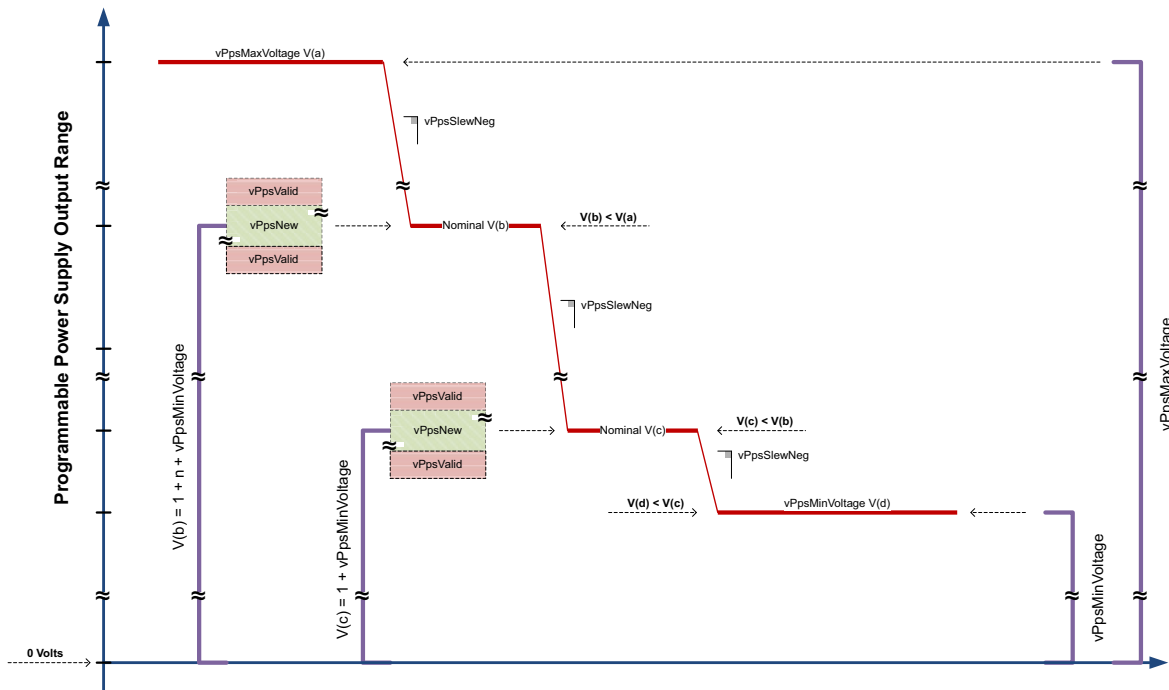


Figure 7.5 PPS Negative Voltage Transitions



[Section 7.1.14, "Non-application of VBUS Slew Rate Limits"](#) lists transitions that are exempt from the **vPpsSlewNeg** and **vPpsSlewPos** limits.

See [Section 7.1.8.1, "AVS/PPS Output Voltage Ripple"](#) for output voltage ripple limits.

See [Section 7.1.8.2, "AVS/PPS DNL Errors and Output Voltage/Current Tolerance"](#) for output voltage and current **DNL** step adjustments.

7.1.4.2.2 SPR Programmable Power Supply Current Limit

The Programmable Power Supply operating in **SPR PPS Mode** **Shall** limit its output current to the **Operating Current** field value in the **RDO** when the **Sink** attempts to draw more current than the **Operating Current** field value level. The programming step size for the Operating Current is **iPpsCLStep**. All programming changes of the Operating Current **Shall** settle to the new **Operating Current** field value within **tPpsCLProgramSettle**. The **SPR PPS** Operating Current regulation accuracy during **Current Limit** is defined as **iPpsCLNew**. The minimum programmable **Current Limit** level is **iPpsCLMin**. A **Source** that supports **SPR PPS Mode** **Shall** support **Current Limit** programmability between **iPpsCLMin** and the Maximum Current value in the **SPR PPS APDO**. A **Source** which receives a request for current below **iPpsCLMin** **Should** reject the request. A **Source** that accepts a request for current below **iPpsCLMin** **Shall** set its current limit at 1A.

The response of an **SPR PPS** to a load change depends on the Operating mode of the **SPR PPS** and the magnitude of the load change. These dependencies lead to one of four possible responses of an **SPR PPS** to any load change. They are differentiated by the value of the PPS Status OMF before and after the load change:

- If the PPS Status OMF is cleared both before and after the load change, the **SPR PPS** responds solely by maintaining the output voltage. The **SPR PPS** output voltage **Shall** remain within **vPpsValid** range. The **SPR PPS** response to the load change **Shall** settle within the **vPpsNew** tolerance band by the time **tPpsTransient**. The Operating Mode Flag **Shall** remain cleared during the load change response of the **SPR PPS**.
- If the PPS Status OMF is cleared before the load change and set after the load change, the **SPR PPS** responds by reducing its output voltage to limit the **SPR PPS** output current. The **SPR PPS** output current **Shall** stay within the **iPpsCVCLTransient** range once it reaches the **iPpsCVCLTransient** range. The **SPR**

PPS response to the load change **Shall** settle within the *iPpsCLNew* tolerance band by the time *tPpsCVCLTransient*. The Operating Mode Flag **Shall** be set when the SPR PPS load change response settles.

- If the PPS Status OMF is set both before and after the load change, the SPR PPS responds by adjusting its output voltage to maintain the output current. The SPR PPS output current **Shall** stay within the *iPpsCLTransient* range. The SPR PPS response to the load change **Shall** settle within the *iPpsCLNew* tolerance band by the time *tPpsCLSettle*. The Operating Mode Flag **Shall** remain set during the load change response of the SPR PPS.
- If the PPS Status OMF is set before the load change and cleared after the load change, the PPS responds to the load change by increasing its output voltage to *vPpsNew* and then maintaining it. The SPR PPS output voltage **Shall** stay within the *vPpsCLCVTransient* range. The SPR PPS response to the load change **Shall** settle within the *vPpsNew* tolerance band by the time *tPpsCLCVTransient*. The Operating Mode Flag **Shall** be cleared when the PPS load change response settles.

The SPR PPS Source **Shall** maintain its output voltage at the value requested in the PPS RDO for all static and dynamic load conditions except when in *Current Limit* operation. In response to any static or dynamic load condition during *Current Limit* operation that causes the SPR PPS output voltage to drop below *vPpsShutdown* the Source **May** send *Hard Reset* Signaling and **Shall** discharge *VBUS* to *vSafe0V* then resumes USB Default Operation at *vSafe5V*.

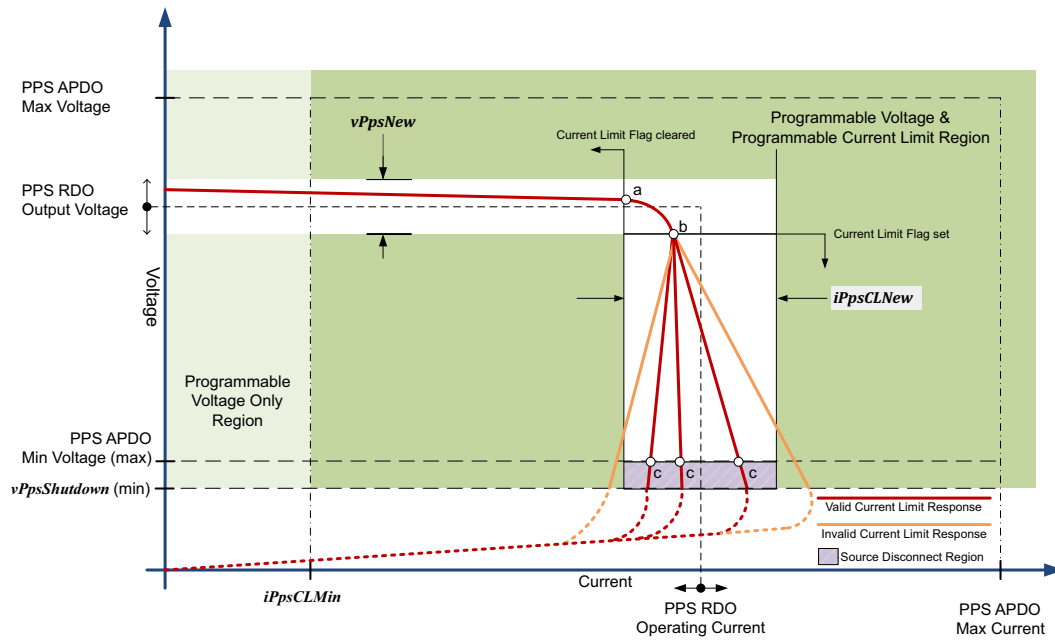
When the Sink attempts to draw more current than the Operating Current in the RDO, the Source **Shall** limit its output current. The current available from the Source during *Current Limit* mode **Shall** meet *iPpsCLNew*. The Sink **May Not** reduce its Operating Current request in the RDO when the PPS Status OMF is set.

Current limiting **Shall** be performed by the SPR PPS Source. Sinks that rely on PPS Current Limiting **Shall** meet the requirements of [Section 7.2.9, "Robust Sink Operation"](#). The Source **Shall Not** shutdown or otherwise disrupt the available output power while in *Current Limit* mode unless another protection mechanism as outlined in [Section 7.1.7, "Robust Source Operation"](#) is engaged to protect the Source from damage.

An SPR PPS Source that is operating in *Current Limit* **Shall Not** change its set-point in a manner that exceeds *iPpsCLLoadStepRate* or *iPpsCLLoadReleaseRate*.

The relationship between SPR PPS programmable output voltage and SPR PPS programmable *Current Limit* **Shall** be as shown in [Figure 7.6, "SPR PPS Programmable Voltage and Current Limit"](#). The transition between the *Constant Voltage* mode and the *Current Limit* mode occurs between points a and b. The PPS Status OMF **Shall** be set or cleared within this region. In *Current Limit* mode when the load resistance changes, the output current of the Source **Shall** stay within *iPpsCLNew*. The proper behavior is represented by point c.

Figure 7.6 SPR PPS Programmable Voltage and Current Limit



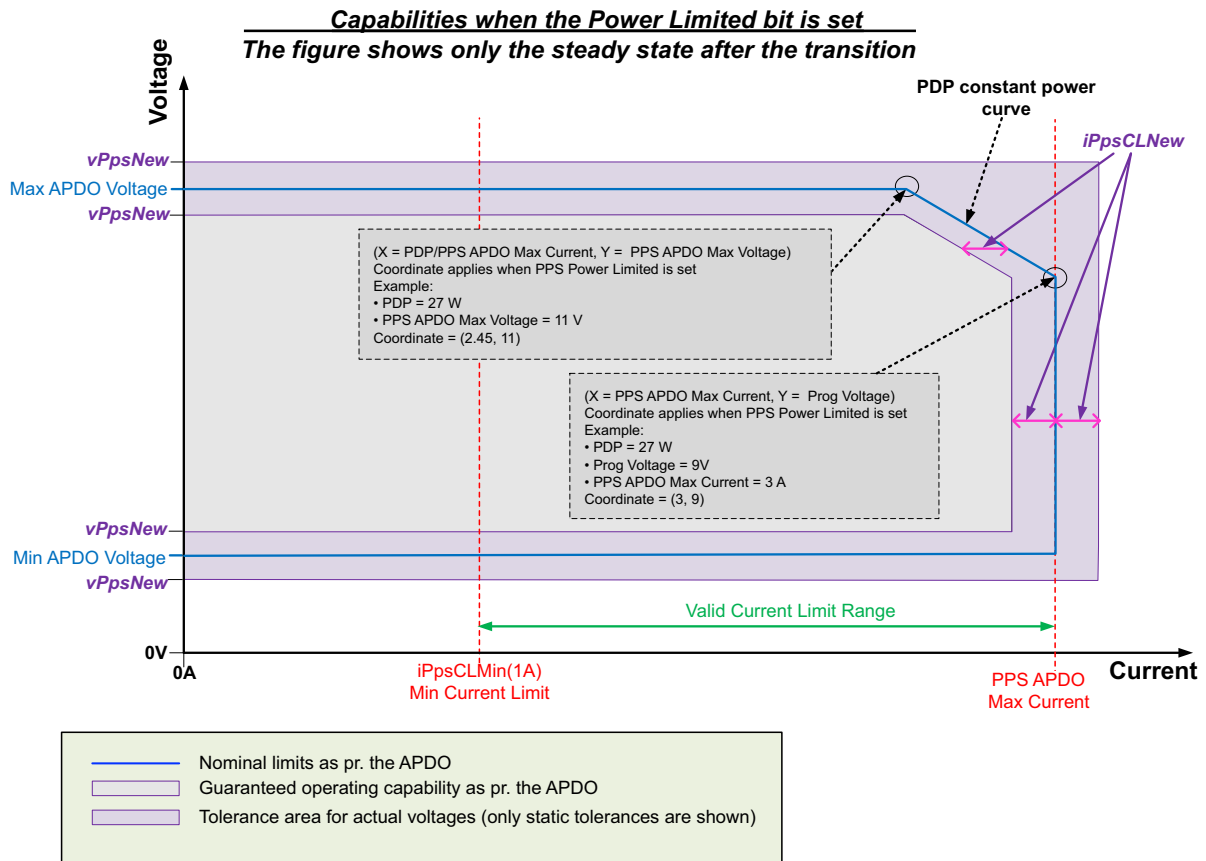
Point *a* represents entry into the transition region between Constant Voltage mode and Current Limit mode.
 Point *b* represents exit from the transition region between Constant Voltage mode and Current Limit mode.
 Point *c* represents the exit from the *iPpsCLNew* region as the voltage drops below the PPS APDO Min Voltage.
 The Source **May** disconnect at any point inside the tolerance range of the minimum voltage defined in the PPS APDO.

7.1.4.2.3 SPR PPS Constant Power Mode

In Constant Power mode (when the PPS Power Limited bit is set) the Source **May** supply power that exceeds the Source's *PDP Rating*. Sinks **May** limit their Operating Current request in the *RDO* and **Shall** meet the requirements of [Section 7.2.9, "Robust Sink Operation"](#).

The tolerances along the Constant Power Curve **Shall Not** extend into the Guaranteed Capability Area of [Figure 7.7, "SPR PPS Constant Power"](#).

Figure 7.7 SPR PPS Constant Power



7.1.4.3 Adjustable Voltage Supply (AVS)

7.1.4.3.1 Adjustable Voltage Supply Voltage Transitions

The *Adjustable Voltage Supply (AVS)* **Shall** transition V_{BUS} over the defined voltage range in a controlled manner. The Output Voltage value in the AVS RDO defines the nominal value of the AVS output voltage after completing a voltage change and **Shall** settle within the limits defined by v_{AvsNew} by $t_{AvsSrcTransSmall}$ for steps smaller than or equal to $v_{AvsSmallStep}$, or else, within the limits defined by v_{AvsNew} by $t_{AvsSrcTransLarge}$ for steps larger than $v_{AvsSmallStep}$. Any overshoot beyond v_{AvsNew} **Shall Not** exceed $v_{AvsValid}$ at any time. Any undershoot beyond v_{AvsNew} **Shall Not** exceed $v_{AvsValid}$ at any time. The AVS output voltage **May** change in a stepwise or linear manner and the slew rate of either type of change **Shall Not** exceed $v_{AvsSlewPos}$ for voltage increases or $v_{AvsSlewNeg}$ for voltage decreases. The nominal requested voltage of all linear voltage changes **Shall** equate to an integer number of *LSB* changes. An *LSB* change of the AVS output voltage is defined as $v_{AvsStep}$. An AVS **Shall** be able to supply the *Negotiated* current level as it changes its output voltage to the requested level if the change of output voltage is less than or equal to $v_{AvsSmallStep}$ relative to v_{AvsNew} . All AVS voltage increases **Shall** result in a voltage that is greater than or equal to the previous AVS output voltage. Likewise, all AVS voltage decreases **Shall** result in a voltage that is less than or equal to the previous AVS output voltage. Any time the *Source* enters the AVS range of operation that voltage transition is considered a voltage step larger than $v_{AvsSmallStep}$.

When the AVS voltage steps up or down, a PS_RDY Message **Shall** be sent within:

- $t_{AvsSrcTransLarge}$ after the last bit of the *GoodCRC* Message following the *Accept* Message for steps larger than $v_{AvsSmallStep}$.
- $t_{AvsSrcTransSmall}$ after the last bit of the *GoodCRC* Message following the *Accept* Message for steps less than or equal to $v_{AvsSmallStep}$ provided the voltage on V_{BUS} has reached v_{AvsNew} .

[Figure 7.8, "AVS Positive Voltage Transitions"](#) and [Figure 7.9, "AVS Negative Voltage Transitions"](#) below show the output voltage behavior of an AVS in response to positive and negative voltage change requests. The parameters $v_{AvsMinVoltage}$ and $v_{AvsMaxVoltage}$ define the lower and upper limits of the AVS range respectively:

- For *SPR* AVS Sources there are two possible voltage ranges where the $v_{AvsMinVoltage}$ is always 9V and $v_{AvsMaxVoltage}$ is either 15V or 20V depending on the *Source's* PDP. See [Table 10.9, "SPR Adjustable Voltage Supply \(AVS\) Voltage Ranges"](#).
- For *EPR* AVS Sources $v_{AvsMinVoltage}$ corresponds to *Minimum Voltage* field (always 15V) in the *EPR* AVS APDO and $v_{AvsMaxVoltage}$ corresponds to *Maximum Voltage* field in the *EPR* AVS APDO. See [Table 10.15, "EPR Adjustable Voltage Supply \(AVS\) Voltage Ranges"](#) for required ranges.

Figure 7.8 AVS Positive Voltage Transitions

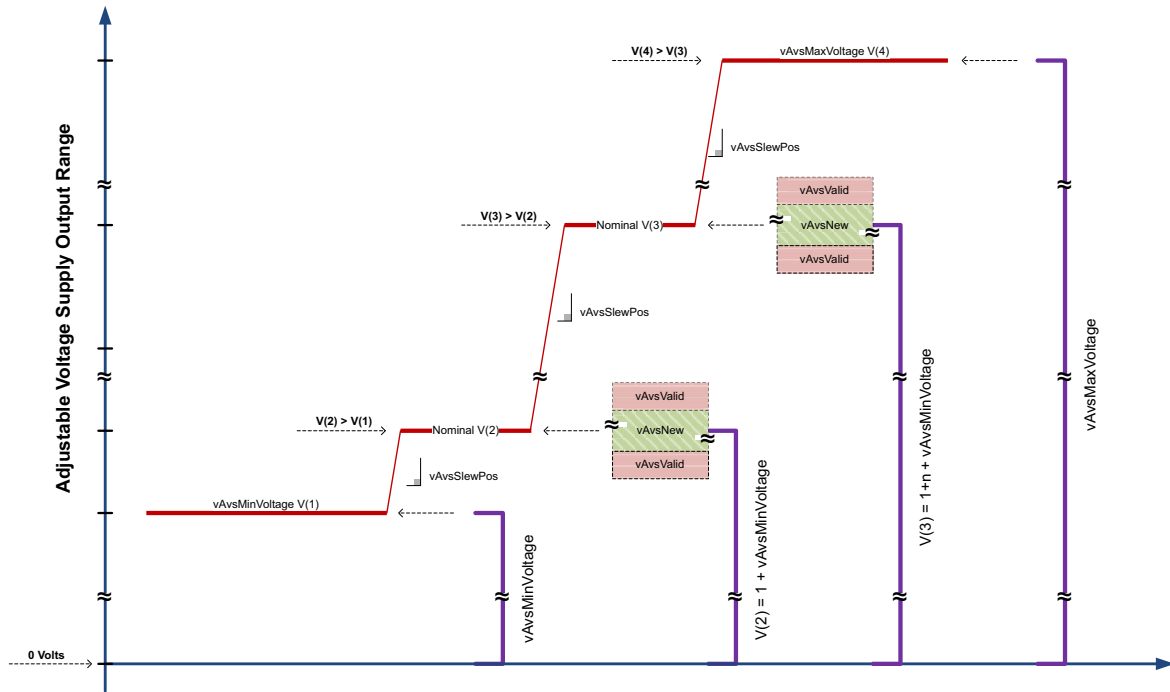
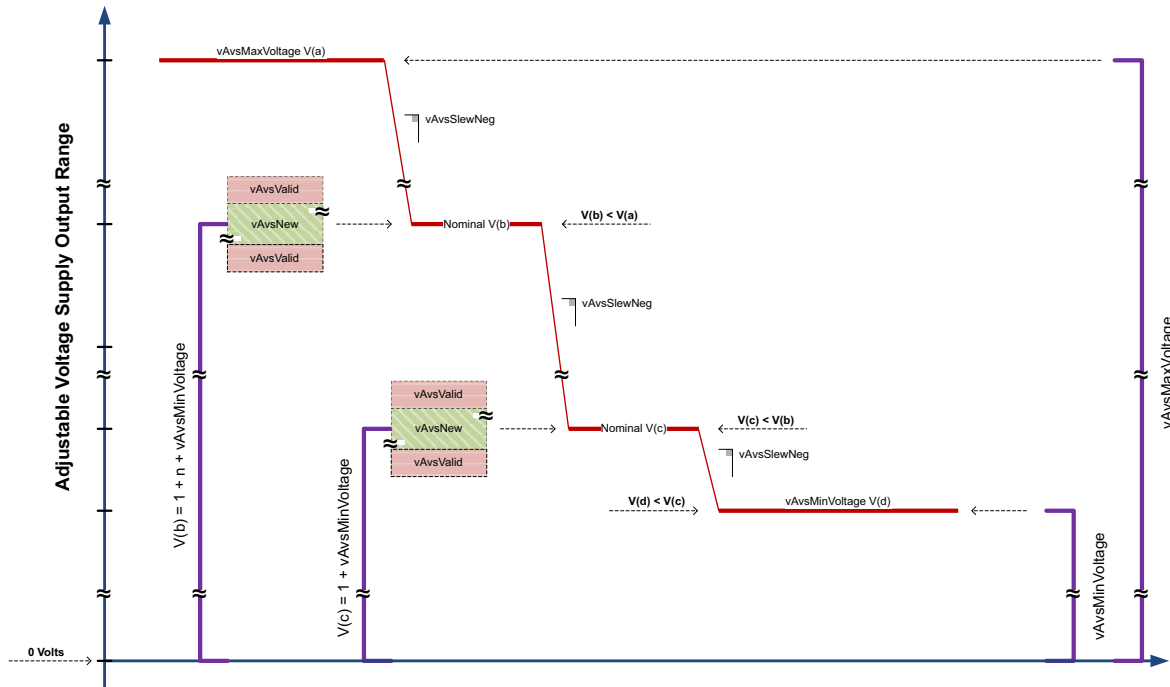


Figure 7.9 AVS Negative Voltage Transitions



See [Section 7.1.8.1, "AVS/PPS Output Voltage Ripple"](#) for output voltage ripple limits.

See [Section 7.1.8.2, "AVS/PPS DNL Errors and Output Voltage/Current Tolerance"](#) for output voltage **DNL** step adjustments.

7.1.4.3.2 Adjustable Voltage Supply Current

The AVS **shall** maintain its output voltage at the value requested in the AVS RDO for all static and dynamic load conditions that do not exceed the Operating Current in the RDO. Unlike the SPR PPS programmable current, the AVS programmable power **may** range from zero to the PDP.

The maximum operating current:

- For SPR Sources, the maximum operating current is defined in the SPR *Source_Capabilities* Message *Maximum Current 15V/Maximum Current 20V* fields.
- For EPR Sources, the maximum operating current has to be calculated as the lower of the *PDP* field value/*Output Voltage* or 5A whichever is lower. See [Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable"](#)

7.1.5 Response to Hard Resets

Hard Reset Signaling indicates a communication failure has occurred and the Source **shall** stop driving *VCONN*, **shall** remove R_p from the *VCONN* pin and **shall** drive *VBUS* to *vSafe0V* as shown in [Figure 7.10, "Source VBUS and VCONN Response to Hard Reset"](#). The USB connection **may** reset during a Hard Reset since the *VBUS* voltage will be less than *vSafe5V* for an extended period of time. After establishing the *vSafe0V* voltage condition on *VBUS*, the Source **shall** wait *tSrcRecover* before re-applying *VCONN* and restoring *VBUS* to *vSafe5V*. A Source **shall** conform to the *VCONN* timing as specified in [\[USB Type-C 2.4\]](#).

A Sink that enters Hard Reset can have *cSnkBulkPd* present until *VBUS* drops below *vSafe0V*. The Source **shall** take this into consideration.

Device operation during and after a Hard Reset is defined as follows:

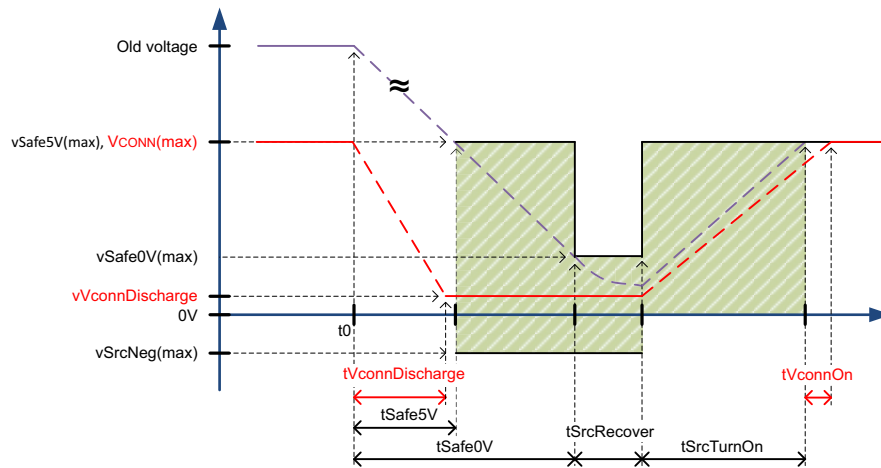
- Self-powered devices **should not** disconnect from USB during a Hard Reset (see [Section 9.1.2, "Mapping to USB Device States"](#)).
- Self-powered devices operating at more than *vSafe5V* **may not** maintain full functionality after a Hard Reset.
- Bus powered devices will disconnect from USB during a Hard Reset due to the loss of their power source.

When a Hard Reset occurs the Source **shall** stop driving *VCONN*, **shall** remove R_p from the *VCONN* pin and **shall** start to transition the *VBUS* voltage to *vSafe0V* either:

- *tPSHardReset* after the last bit of the **Hard Reset** Signaling has been received from the Sink or
- *tPSHardReset* after the last bit of the **Hard Reset** Signaling has been sent by the Source.

The Source **shall** meet both *tSafe5V* and *tSafe0V* relative to the start of the voltage transition as shown in [Figure 7.10, "Source VBUS and VCONN Response to Hard Reset"](#).

Figure 7.10 Source V_{BUS} and V_{CONN} Response to Hard Reset



V_{CONN} will meet $tV_{CONNDischarge}$ relative to the start of the voltage transition as shown in [Figure 7.10, "Source \$V_{BUS}\$ and \$V_{CONN}\$ Response to Hard Reset"](#) due to the discharge circuitry in the *Cable Plug*. V_{CONN} **Shall** meet tV_{CONNOn} relative to V_{BUS} reaching $vSafe5V$.

Note: tV_{CONNOn} and $tV_{CONNDischarge}$ are defined in [\[USB Type-C 2.4\]](#).

7.1.6 Changing the Output Power Capability

Some USB Power Delivery *Negotiations* will require the *Source* to adjust its output power capability without changing the output voltage. In this case the *Source* **Shall** be able to supply a higher or lower load current within $tSrcReady$.

7.1.7 Robust Source Operation

7.1.7.1 Output Over Current Protection

Sources **Shall** implement over current protection to prevent damage from output current that exceeds the current handling capability of the *Source*. The definition of current handling capability is left to the discretion of the *Source* implementation and **Shall** take into consideration the current handling capability of the connector contacts. If the over current protection implementation does not use a *Hard Reset* or *Error Recovery*, it **Shall Not** interfere with the *Negotiated V_{BUS}* current level.

After three consecutive over current events *Source* **Shall** go to *ErrorRecovery*.

Sources **Should** attempt to send *Hard Reset* Signaling when over current protection engages followed by an *Alert Message* indicating an *OCP* event once an *Explicit Contract* has been established. The over current protection response **May** engage at either the *Port* or system level. Systems or ports that have engaged over current protection **Should** attempt to resume *USB Default Operation* after determining that the cause of over current is no longer present and **May** latch off to protect the *Port* or system. The definition of how to detect if the cause of over current is still present is left to the discretion of the *Source* implementation.

The *Source* **Shall** *Re-negotiate* with the *Sink* after choosing to resume *USB Default Operation*. The decision of how to *Re-negotiate* after an over current event is left to the discretion of the *Source* implementation.

The *Source* **Shall** prevent continual system or *Port* cycling if over current protection continues to engage after initially resuming either *USB Default Operation* or *Re-negotiation*. Latching off the *Port* or system is an acceptable response to recurring over current.

During the over current response and subsequent system or *Port* shutdown, all affected *Source* ports operating with V_{BUS} greater than $vSafe5V$ **Shall** discharge V_{BUS} to $vSafe5V$ by the time $tSafe5V$ and $vSafe0V$ by the time $tSafe0V$.

7.1.7.2 Over Temperature Protection

Sources **Shall** implement Over Temperature Protection (OTP) to prevent damage from temperature that exceeds the thermal capability of the *Source*. The definition of thermal capability and the monitoring locations used to trigger the over temperature protection are left to the discretion of the *Source* implementation.

In order to avoid reaching an OTP event, *Sources* **May** proactively reduce the available power being offered to the *Sink*, even though these offers might be lower than the *Source* would be expected to offer during normal thermal operating conditions. Prior to reducing power, the *Source* **Should** generate *Alert Message* indicating an Operating Condition Change and set the Temperature Status bit in the *SOP Status Message* to Warning (10b).

Sources **Should** attempt to send *Hard Reset Signaling* when OTP engages followed by an *Alert Message* indicating an OTP event once an *Explicit Contract* has been established. The OTP response **May** engage at either the *Port* or system level. Systems or ports that have engaged OTP **Should** attempt to resume *USB Default Operation* and **May** latch off to protect the *Port* or system.

The *Source* **Shall** Re-negotiate with the *Sink* after choosing to resume *USB Default Operation*. The decision of how to Re-negotiate after an over temperature event is left to the discretion of the *Source* implementation.

The *Source* **Shall** prevent continual system or *Port* cycling if over temperature protection continues to engage after initially resuming either *USB Default Operation* or *Re-negotiation*. Latching off the *Port* or system is an acceptable response to recurring over temperature.

During the OTP and subsequent system or *Port* shutdown, all affected *Source* ports operating with *VBUS* greater than *vSafe5V* **Shall** discharge *VBUS* to *vSafe5V* by the time *tSafe5V* and *vSafe0V* by the time *tSafe0V*.

7.1.7.3 vSafe5V Externally Applied to Ports Supplying vSafe5V

Safe operation mandates that Power Delivery *Sources* **Shall** be tolerant of *vSafe5V* being present on *VBUS* when simultaneously applying power to *VBUS*. Normal USB PD communication **Shall** be supported when this *vSafe5V* to *vSafe5V* connection exists.

7.1.7.4 Detach

A USB *Detach* is detected electrically using *CC* detection on the *USB Type-C* connector. When the *Source* is *Detached* the *Source* **Shall** transition to *vSafe0V* by *tSafe0V* relative to when the *Detach* event occurred. During the transition to *vSafe0V* the *VBUS* voltage **Shall** be below *vSafe5V* max by *tSafe5V* relative to when the *Detach* event occurred and **Shall Not** exceed *vSafe5V* max after this time.

Sources operating in *EPR Mode* need to avoid creating large differential voltages at the connector. See Appendix H in the [\[USB Type-C 2.4\]](#) specification for background information. To achieve this, *Sources* operating in *EPR Mode*, upon detecting a disconnect, **Shall** stop sourcing current and minimize *VBUS* capacitance. There **May** continue to be current sourced from the *Source* bulk capacitance, but that **Should** also be minimized by disconnecting as much of the *Source* bulk capacitance as possible. For example, the *Source* can stop sourcing from the Power Supply and the C1 portion of the *Source* bulk capacitance in [Figure 7.1, "Placement of Source Bulk Capacitance"](#) by disabling the Ohmic Interconnect switch.

The *Source* **Should** detect the disconnect, stop sourcing current, and minimize the *VBUS* capacitance as quickly as practical. If this is done after the *CC* contacts disconnect and before the *VBUS* contacts disconnect there is less risk of large differential voltages at the connector.

Note: A USB-PD transmission by the *Source* during a disconnect event will delay disconnect detection by the *Source*.

7.1.7.5 Output Voltage Limit

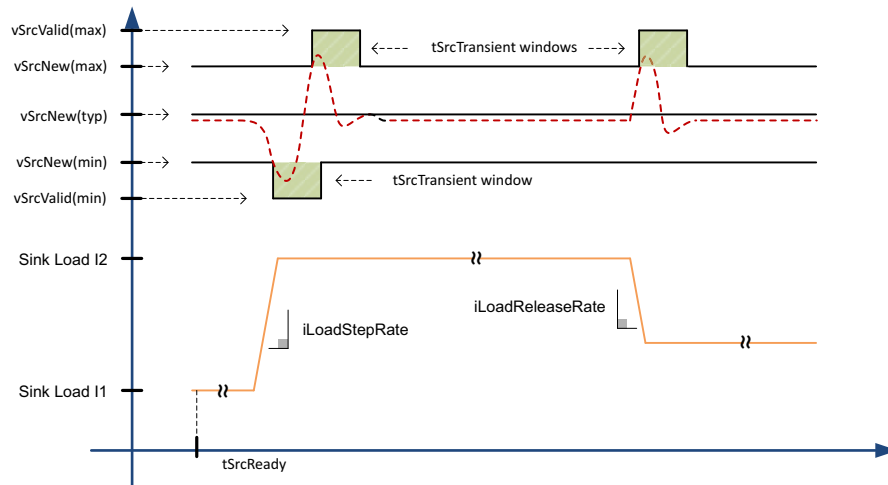
The output voltage of *Sources* **Shall** account for *vSrcNew*, *vSrcValid* or *vPpsNew*, *vPpsValid* or *vAvsNew*, *vAvsValid* as determined by the *Negotiated VBUS* value. *Sources* **Shall** meet applicable safety and regulatory requirements.

7.1.8 Output Voltage Tolerance and Range

After a voltage transition is complete (i.e., after *tSrcReady*) and during static load conditions the *Source* output voltage **Shall** remain within the *vSrcNew* or *vSafe5V* limits as applicable. The ranges defined by *vSrcNew* and *vSafe5V* account for DC regulation accuracy, line regulation, load regulation and output ripple. After a voltage transition is complete (i.e., after *tSrcReady*) and during transient load conditions the *Source* output voltage **Shall Not** go beyond the range specified by *vSrcValid*. The amount of time the *Source* output voltage can be in the band between either *vSrcNew* or *vSafe5V* and *vSrcValid* **Shall Not** exceed *tSrcTransient*. Refer to [Table 7.23, "Source Electrical Parameters"](#) for the output voltage tolerance specifications. [Figure 7.11, "Application of vSrcNew and vSrcValid limits after tSrcReady"](#) illustrates the application of *vSrcNew* and *vSrcValid* after the voltage transition is complete.

The *vSrcNew* and *vSrcValid* limits **Shall Not** apply to *VBUS* during the *VBUS* discharge and switchover that occurs during a *Fast Role Swap* as described in [Section 7.1.13, "Fast Role Swap"](#).

Figure 7.11 Application of vSrcNew and vSrcValid limits after tSrcReady

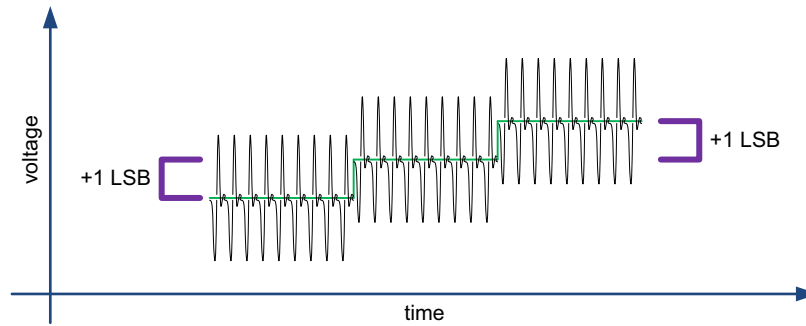


The *Source* output voltage **Shall** be measured at the connector receptacle. The stability of the *Source* **Shall** be tested in 25% load step increments from minimum load to maximum load and also from maximum load to minimum load. The transient behavior of the load current is defined in [Section 7.2.6, "Transient Load Behavior"](#). The time between each step **Shall** be sufficient to allow for the output voltage to settle between load steps. In some systems it might be necessary to design the *Source* to compensate for the voltage drop between the output stage of the power supply electronics and the receptacle contact. The determination of whether compensation is necessary is left to the discretion of the *Source* implementation.

7.1.8.1 AVS/PPS Output Voltage Ripple

The AVS/PPS output voltage ripple is expected to exceed the magnitude of one or more *LSB* as show in the [Figure 7.12, "Expected AVS/PPS Ripple Relative to an LSB"](#).

Figure 7.12 Expected AVS/PPS Ripple Relative to an LSB



7.1.8.2 AVS/PPS DNL Errors and Output Voltage/Current Tolerance

The PPS voltage and current discrete *LSB* steps have a **DNL** tolerance as shown in [Figure 7.13, "Allowed DNL errors and tolerance of Voltage and Current in AVS/PPS mode"](#) below. In absolute terms the step size of the *LSB* for both voltage and current is defined by **vPpsStep/vAvsStep** for voltage and **iPpsCLStep** for current. Several examples of **Valid** *LSB* steps are shown in [Figure 7.13, "Allowed DNL errors and tolerance of Voltage and Current in AVS/PPS mode"](#):

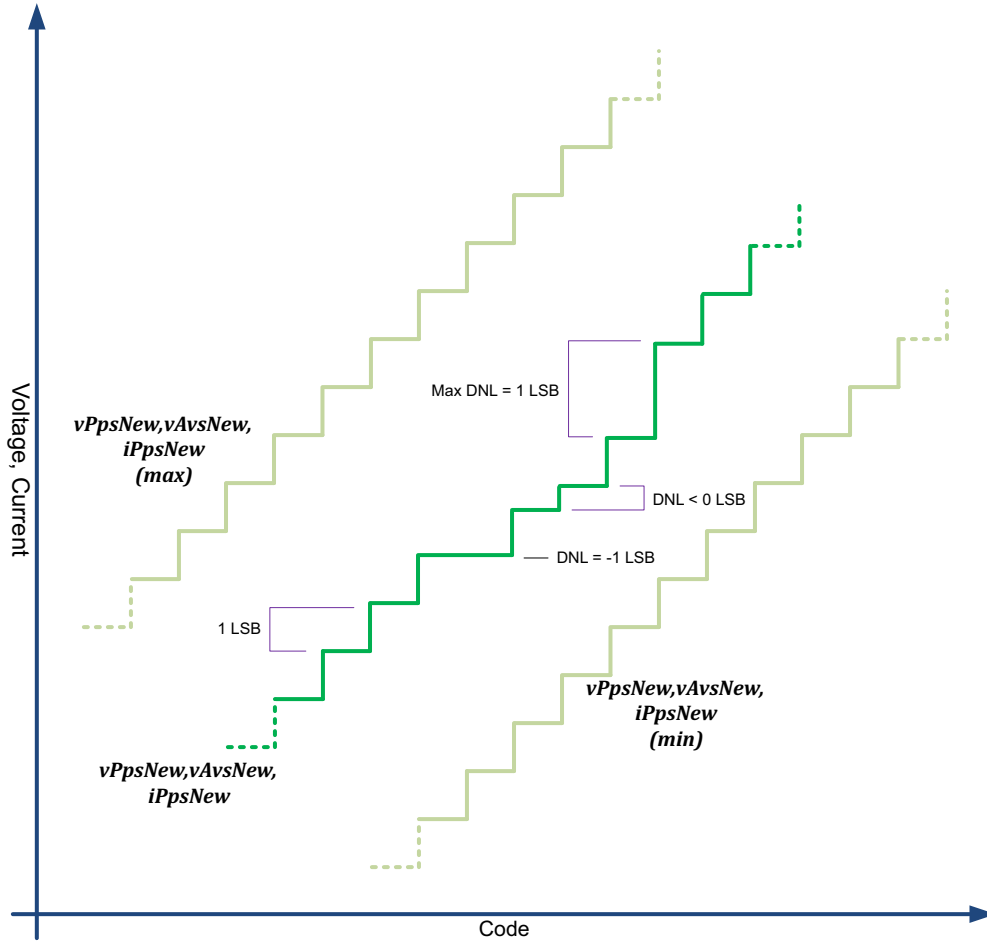
- The upper end of the **DNL** error (+1 *LSB*) shows the case where one step is effectively skipped.
- The lower end of the **DNL** error (-1 *LSB*) shows the case where the voltage or current set-point remained the same.

The ideal scenario for the **DNL** error (=0) matches the typical step size for the voltage or current.

The intent of **DNL** is to guarantee that changes to the voltage/current have the correct directionality, and that the maximum step size is clearly defined.

Note: The Source **Should** avoid scenarios where multiple consecutive steps have errors close to the Maximum and Minimum **DNL**.

Figure 7.13 Allowed DNL errors and tolerance of Voltage and Current in AVS/PPS mode



7.1.8.3 Programmable Power Supply Output Voltage Tolerance and Range

After a voltage transition of a Programmable Power Supply is complete (i.e. after *tPpsSrcTransSmall* or *tPpsSrcTransLarge*) and during static load conditions the *Source* output voltage **Shall** remain within the *vPpsNew* limits. The range defined by *vPpsNew* accounts for DC regulation accuracy, line regulation, load regulation and output ripple. After a voltage transition is complete (i.e. after *tPpsSrcTransSmall* or *tPpsSrcTransLarge*) and during transient load conditions the *Source* output voltage **Shall Not** go beyond the range specified by *vPpsValid*. The amount of time the *Source* output voltage can be in the band between *vPpsNew* and *vPpsValid* **Shall Not** exceed *tPpsTransient*.

7.1.8.4 Adjustable Voltage Supply Output Voltage tolerance and Range

After a voltage transition of an AVS is complete (i.e. after *tAvsSrcTransSmall* or *tAvsSrcTransLarge*) and during static load conditions the *Source* output voltage **Shall** remain within the *vAvsNew* limits. The range defined by *vAvsNew* accounts for DC regulation accuracy, line regulation, load regulation and output ripple. After a voltage transition is complete (i.e. after *tAvsSrcTransSmall* or *tAvsSrcTransLarge*) and during transient load conditions the *Source* output voltage **Shall Not** go beyond the range specified by *vAvsValid*. The amount of time the *Source* output voltage can be in the band between *vAvsNew* and *vAvsValid* **Shall Not** exceed *tAvsTransient*.

7.1.9 Charging and Discharging the Bulk Capacitance on VBUS

The *Source* **Shall** charge and discharge the bulk capacitance on *VBUS* whenever the *Source* voltage is *Negotiated* to a different value. The charging or discharging occurs during the voltage transition and **Shall Not** interfere with the *Source*'s ability to meet *tSrcReady*.

7.1.10 Swap Standby for Sources

Sources and *Sinks* of a *Dual-Role Power Port* **Shall** support *Swap Standby*. *Swap Standby* occurs for the *Source* after the *Source* power supply has discharged the bulk capacitance on *VBUS* to *vSafe0V* as part of the *Power Role Swap* transition.

While in *Swap Standby*:

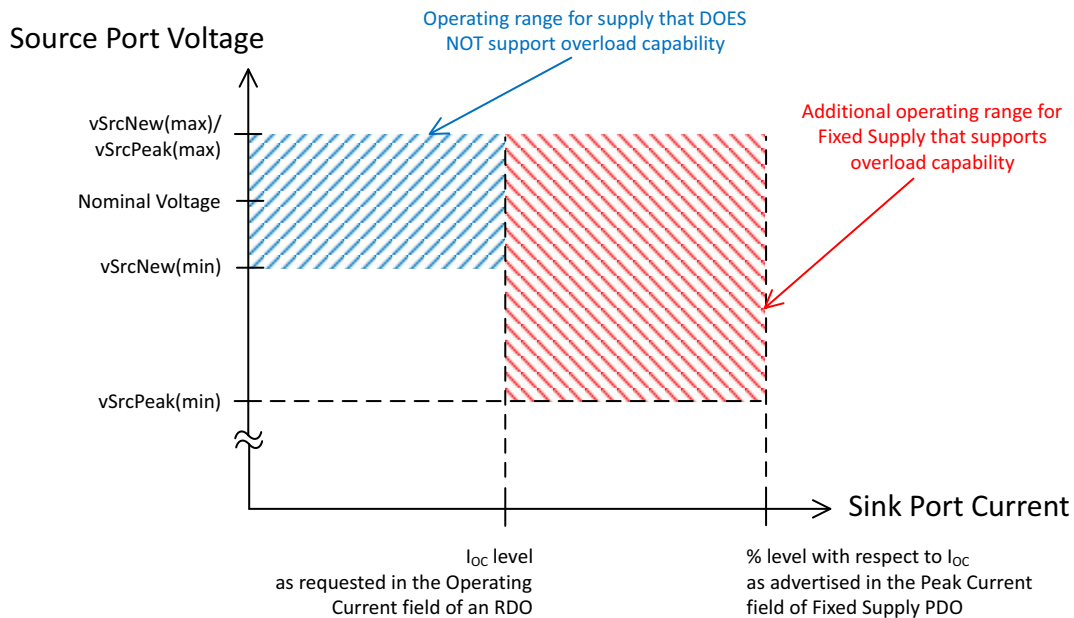
- The *Source* **Shall Not** drive *VBUS* that is therefore expected to remain at *vSafe0V*.
- Any discharge circuitry that was used to achieve *vSafe0V* **Shall** be removed from *VBUS*.
- The *Dual-Role Power Port* **Shall** be configured as a *Sink*.
- The USB connection **Shall Not** reset even though *vSafe5V* is no longer present on *VBUS* (see [Section 9.1.2, "Mapping to USB Device States"](#)).

The *PS_RDY* Message associated with the *Source* being in *Swap Standby* **Shall** be sent after the *VBUS* drive is removed. The time for the *Source* to transition to *Swap Standby* **Shall Not** exceed *tSrcSwapStdby*. Upon entering *Swap Standby*, the *Source* has relinquished its *Power Role* as *Source* and is ready to become the *New Sink*. The transition time from *Swap Standby* to being the *New Sink* **Shall** be no more than *tNewSnk*. The *New Sink* **May** start using power after the new *Source* sends the *PS_RDY* Message.

7.1.11 Source Peak Current Operation

A *Source* that has the *Fixed Supply PDO* or *AVS APDO* Peak Current bits set to 01b, 10b and 11b **Shall** be designed to support one of the overload *Capabilities* defined in [Table 6.10, "Fixed Power Source Peak Current Capability"](#) or [Table 6.16, "EPR AVS Power Source Peak Current Capability"](#) respectively. The overload conditions are bound in magnitude, duration and duty cycle as listed in [Table 6.10, "Fixed Power Source Peak Current Capability"](#) or [Table 6.16, "EPR AVS Power Source Peak Current Capability"](#). *Sources* are not required to support continuous overload operation. When overload conditions occur, the *Source* is allowed the range of *vSrcPeak* (instead of *vSrcNew*) relative to the nominal value (see [Figure 7.14, "Source Peak Current Overload"](#)). When the overload capability is exceeded, the *Source* is expected take whatever action is necessary to prevent electrical or thermal damage to the *Source*. The *Source* **May** send a new *Source_Capabilities* Message with the *Fixed Supply PDO* or *AVS APDO* Peak Current bits set to 00b to prohibit overload operation even if an overload capability was previously *Negotiated* with the *Sink*.

Figure 7.14 Source Peak Current Overload



7.1.12 Source Capabilities Extended Parameters

Implementers can choose to make available certain characteristics of a *PDUSB Source* as a set of **Static** and/or dynamic parameters to improve interoperability between external power sources and portable computing devices. The complete list of reportable **Static** parameters is described in full in [Section 6.5.1, "Source Capabilities Extended Message"](#) and listed in [Figure 6.37, "Source Capabilities Extended Message"](#). The subset of parameters listed below directly represent *Source Capabilities* and are described in the rest of this section.

- Voltage Regulation.
- Holdup Time.
- Compliance.
- Peak Current.
- *Source* Inputs.
- Batteries.

7.1.12.1 Voltage Regulation Field

The power consumption of a device can change dynamically. The ability of the *Source* to regulate its voltage output might be important if the device is sensitive to fluctuations in voltage. The **Voltage Regulation** bit field is used to convey information about the *Sources* output regulation and tolerance to various load steps.

7.1.12.1.1 Load Step Slew Rate

The default load step slew rate is established at 150mA/μs. A *Source* **Shall** meet the following requirements under the load step reported in the **Source Capabilities Extended Message**:

- The *Source* **Shall** maintain *V_{BUS}* regulation within the **vSrcValid** range.
- The noise on the *CC* line **Shall** remain below **vNoiseIdle** and **vNoiseActive**.

Test conditions require a change in both positive and negative load steps from 1Hz to 5000Hz, up to the *Advertised Load Step Magnitude* of the full load output including from both 10 mA and 10% initial load. The *Source* **Shall** ensure that PD Communications meet the transmit and receive masks as specified in [Section 5.8.2, "Transmit and Receive Masks"](#) under all load conditions.

7.1.12.1.2 Load Step Magnitude

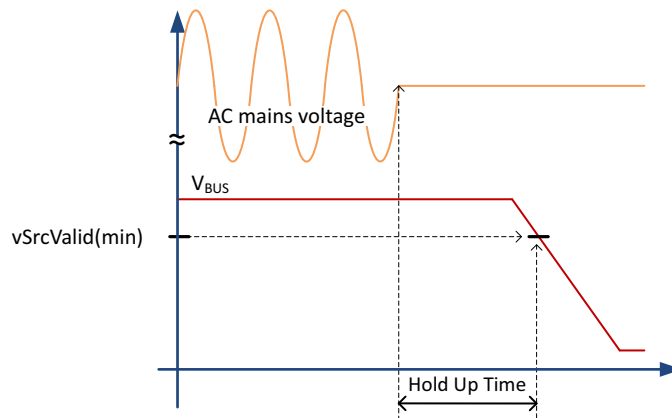
The default load step magnitude rate **Shall** be 25% of I_{OC} . The *Source* **May** report higher capability tolerating a load step of 90% of I_{OC} .

7.1.12.2 Holdup Time Field

The *Holdup Time* field **Shall** return a numeric value of the number of milliseconds the output voltage stays in regulation upon a short interruption of the *AC Supply*.

An *AC Supplied Source* **Shall** report its holdup time in this field. The holdup time is measured with the load at rated maximum, with the *AC Supply* at 115VAC rms and 60Hz (or at 230VAC rms and 50Hz for a *Source* that does not support 115VAC *AC Supply*). The reported time describes the minimum length of time from the last completed *AC Supply* input cycle (zero-degree phase angle) until when the output voltage decays below *vSrcValid* (min). *Sources* are recommended to support a minimum of 3ms and are preferred to support over 10 milliseconds holdup time (equivalent to a half cycle drop from the *AC Supply*). See [Figure 7.15, "Holdup Time Measurement"](#).

Figure 7.15 Holdup Time Measurement



7.1.12.3 Compliance Field

An *SPR Source* claiming *LPS*, PS1 or PS2 compliance (see [\[IEC 62368-1\]](#)) **Shall** report its *Capabilities* in the *Compliance* field. Since the *SPR Source* **May** have several potential output voltage and current settings, every *SPR Source* supply (each indicated by a *PDO*) **Shall** be compliant to *LPS* requirements.

Note: According to the requirements of [\[IEC 60950-1\]](#) and/or [\[IEC 62368-3\]](#), a device tested and certified with an *LPS Source* (*SPR Source* or *EPR Source* operating in *SPR Mode*) is prohibited from using a non-*LPS Source* (*EPR Source* operating in *EPR Mode*). Alternatively, [\[IEC 62368-1\]](#), classifies power sources according to their maximum, constrained power output (15watts or 100watts).

7.1.12.4 Peak Current

The *Source* reports its ability to source peak current delivery in excess of the *Negotiated* amount in the *Peak Current* field. The duration of peak current **Shall** be followed by a current consumption below the Operating Current (I_{OC}) in order to maintain average power delivery below the I_{OC} current.

A Source **May** have greater capability to source peak current than can be reported using the **Peak Current** field in the *Fixed Supply PDO* or *AVS APDO*. In this case the Source **Shall** report its additional capability in the **Peak Current1/Peak Current2/Peak Current3** fields in the *Source_Capabilities_Extended* Message.

Each overload period **Shall** be followed by a period of reduced current draw such that the rolling average current over the Overload Period field value with the specified Duty Cycle field value (see [Section 6.5.1.10, "Peak Current Field"](#)) **Shall Not** exceed the *Negotiated* current. This is calculated as:

Period of reduced current = (1 - value in Duty Cycle field/100) * value in Overload Period field

7.1.12.5 Source Inputs

The **Source Inputs** field identifies the possible inputs that provide power to the Source.

Note: Some Sources are only powered by a Battery (e.g., an automobile) rather than the more common AC Supply.

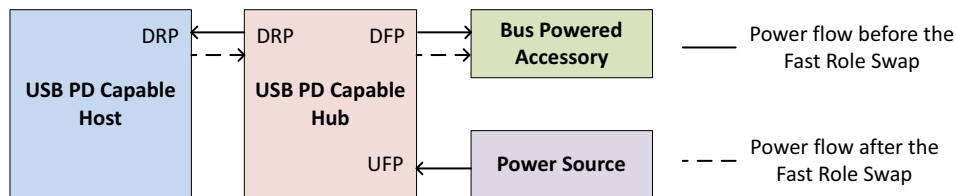
7.1.12.6 Batteries

The **Number of Batteries/Battery Slots** field **Shall** report the number of Batteries the Source supports. The Source **Shall** independently report the number of Hot Swappable Batteries and the number of *Fixed Batteries*.

7.1.13 Fast Role Swap

A *Fast Role Swap* limits the interruption of *VBUS* power to a bus powered accessory connected to a *Hub DFP* that has a *UFP Attached* to a power source and a *DRP Attached* to a *Host Port* supporting *DRP* as shown in [Figure 7.16, "VBUS Power during Fast Role Swap"](#).

Figure 7.16 VBUS Power during Fast Role Swap



When the power source connected to the *Hub UFP* stops sourcing power and *VBUS* at the *Hub DRP* connector discharges below **vSrcValid**(min), if *VBUS* has been *Negotiated* to a higher voltage than **vSafe5V**, or **vSafe5V** (min) the *Fast Role Swap Request* **Shall** be sent from the *Hub DRP* to the *Host DRP* and the *Hub DRP* **Shall** sink power. In the *Fast Role Swap* use case, the *Hub DRP* behaves like a bidirectional power path. The *Hub DRP* **Shall Not** enable *VBUS* discharge circuitry when changing operation from *Initial Source* to *New Sink*. The *Hub DFP* Port(s) **Shall** support default *USB Type-C* Current (see [\[USB Type-C 2.4\]](#)) until a new *Explicit Contract* is *Negotiated*.

After sending the *Fast Role Swap Request* and while *VBUS* > **vSafe5V** (min), the *New Sink* **Shall Not** draw more than **iNewFrsSink** until the *New Source* has applied its R_p . The *New Sink* **Shall Not** draw more than **iSnkStdby** from *VBUS* until **tSnkFRSwap** after it has started sending the *Fast Role Swap Request* or *VBUS* has fallen below **vSafe5V** (min). The **tSnkFRSwap** time **Shall** start at the beginning of the *Fast Role Swap Request* or when *VBUS* falls below **vSafe5V** (min), whichever comes later. After waiting for **tSnkFRSwap**, the *New Sink* **Shall Not** draw more than **iNewFrsSink** until the *New Source* has applied its R_p . After the *New Source* has applied its R_p , the *New Sink* **Shall** be limited to *USB Type-C* Current (see [\[USB Type-C 2.4\]](#)) in an *Implicit Contract* until a new *Explicit Contract* is *Negotiated*. All Sink requirements **Shall** apply to the *New Sink* after the *Fast Role Swap* is complete. The *Fast Role Swap* response of the *Host DRP* is described in [Section 7.2.10, "Fast Role Swap"](#) since the *Host DRP* is operating as the *Initial Sink* prior to the *Fast Role Swap*.

After the *VBUS* voltage level at the *Hub DRP* connector drops below **vSafe5V** a **PS_RDY** Message **Shall** be sent to the *Host DRP* as shown in the *Fast Role Swap* transition diagram of [Section 7.3.4, "Transitions Caused by Fast Role Swap"](#).

[Figure 7.17, "VBUS detection and timing during Fast Role Swap, initial VBUS \(at new source\) > vSafe5V\(min\)"](#) and [Figure 7.18, "VBUS detection and timing during Fast Role Swap, initial VBUS \(at new source\) < vSafe5V\(min\)"](#) show the VBUS detection and timing for the New Source during a Fast Role Swap after the Fast Role Swap Request has been received. The New Source **May** turn on the VBUS output switch once VBUS is below vSafe5V (max). In this case, the New Source prevents VBUS from falling below vSafe5V (min). The new source **Shall** turn on the VBUS output switch within tSrcFRSwap of falling below vSafe5V (min).

VBUS might have started at vSafe5V or at higher voltage. When the Fast Role Swap Request is detected, VBUS could therefore be either above vSafe5V (max), within the vSafe5V range, or below vSafe5V (min). If the Fast Role Swap Request is detected when VBUS is below vSafe5V (min), then the new source **Shall** turn on the VBUS output switch within tSrcFRSwap of detecting the Fast Role Swap Request. In this case, the maximum time from the beginning of the Fast Role Swap Request to VBUS being sourced **May** be tSrcFRSwap (max) + tFRSwapRx (max).

Figure 7.17 VBUS detection and timing during Fast Role Swap, initial VBUS (at new source) > vSafe5V(min)

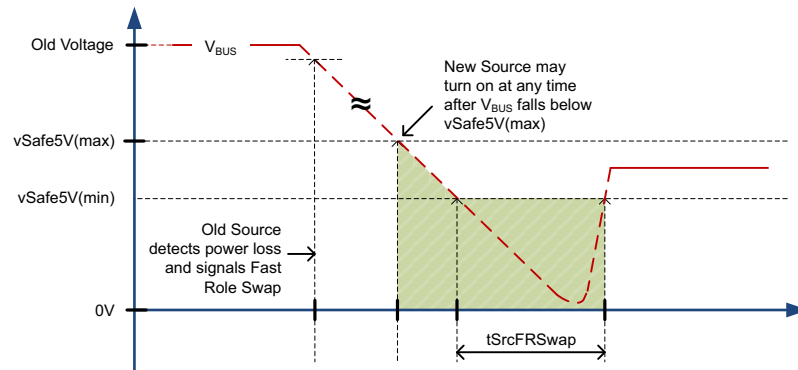
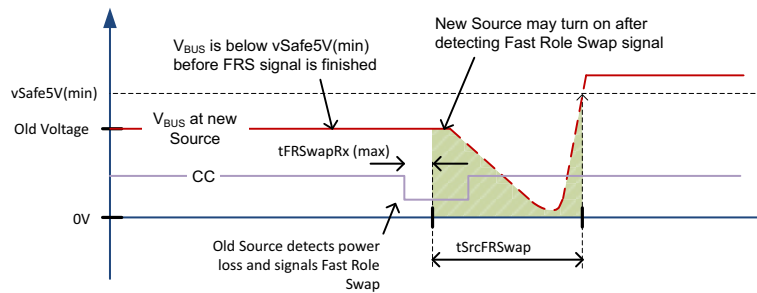


Figure 7.18 VBUS detection and timing during Fast Role Swap, initial VBUS (at new source) < vSafe5V(min)



7.1.14 Non-application of VBUS Slew Rate Limits

Scenarios where vSrcSlewPos and vPpsSlewPos VBUS slew rate limits do not apply and VBUS **May** transition faster than specified are as follows:

- When first applying VBUS after an Attach.
- When applying VBUS as part of a Power Role Swap to Source Power Role.
- When increasing VBUS from vSafe0V to vSafe5V during a Hard Reset.
- During a Fast Role Swap when the Initial Sink applies VBUS.

Scenarios where *vSrcSlewNeg* and *vPpsSlewNeg* *VBUS* slew rate limits do not apply and *VBUS* **May** transition faster than specified are as follows:

- When discharging *VBUS* to *vSafe0V* during a *Hard Reset*.
- When discharging *VBUS* to *vSafe0V* as part of a *Power Role Swap* to *Sink Power Role*.
- When discharging *VBUS* to *vSafe0V* after a *Detach*.
- During a *Fast Role Swap* when the *VBUS* power source connected to the *Hub UFP* stops sourcing power.

7.1.15 VCONN Power Cycle

7.1.15.1 UFP VCONN Power Cycle

The *Data Reset* process requires the *DFP* to be the *VCONN Source* by the end of the process. In the case where the *UFP* is the *VCONN Source*, the following steps **Shall** be followed:

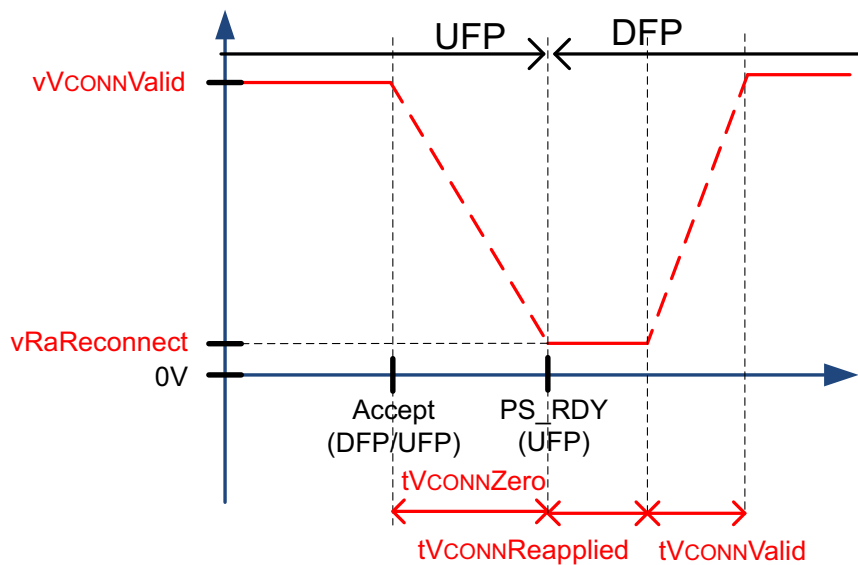
- Following the last bit of the *GoodCRC* Message acknowledging the *Accept* Message in response to the *Data_Reset* Message, the *UFP Shall* turn off *VCONN* and ensure it is below *vRaReconnect* (see [USB Type-C 2.4]) within *tVCONNZero*.
- When *VCONN* is below *vRaReconnect*, the *UFP Shall* send a *PS_RDY* Message.

Note: If the *UFP* was not sourcing *VCONN*, it still sends the *PS_RDY* Message.

- The *DFP Shall* wait *tVCONNReapplied* following the last bit of the *GoodCRC* Message acknowledging the *PS_RDY* Message before sourcing *VCONN*. The *DFP Shall* ensure *VCONN* is within *vVCONNValid* (see [USB Type-C 2.4]) within *tVCONNValid*.

Figure 7.19, "*Data Reset UFP VCONN Power Cycle*" below illustrates the *UFP VCONN* Power Cycle process.

Figure 7.19 Data Reset UFP VCONN Power Cycle



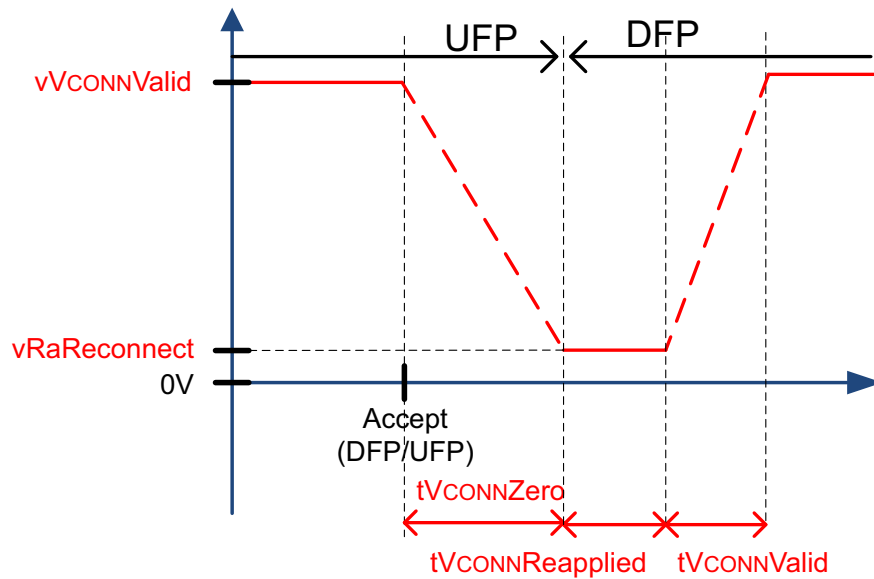
7.1.15.2 DFP VCONN Power Cycle

The *Data Reset* process requires the *DFP* to be the *VCONN Source* by the end of the process. In the case where the *DFP* is the *VCONN Source*, the following steps **Shall** be followed:

- 1) If the *DFP* sent the **Data_Reset** Message and is sourcing *VCONN* then it **Shall** turn off *VCONN* and ensure it is below *vRaReconnect* (see [USB Type-C 2.4]) within *tVCONNZero* of the last bit of the **GoodCRC** Message acknowledging the **Accept** Message in response to the **Data_Reset** Message.
- 2) If the *UFP* sent the **Data_Reset** Message then the *DFP* **Shall** turn off *VCONN* and ensure it is below *vRaReconnect* (see [USB Type-C 2.4]) within *tVCONNZero* following the last bit of the **GoodCRC** Message acknowledging the **Accept** Message in response to the **Data_Reset** Message.
- 3) When *VCONN* is below *vRaReconnect*, the *DFP* **Shall** wait *tVCONNReapplied* before sourcing *VCONN*.
- 4) The *DFP* **Shall** ensure *VCONN* is within *vVCONNValid* (see [USB Type-C 2.4]) within *tVCONNValid*.

Figure 7.20, "*Data Reset DFP VCONN Power Cycle*" below illustrates the *DFP VCONN* Power Cycle process.

Figure 7.20 Data Reset DFP VCONN Power Cycle



7.2 Sink Requirements

7.2.1 Behavioral Aspects

A PDUSB Sink exhibits the following behaviors:

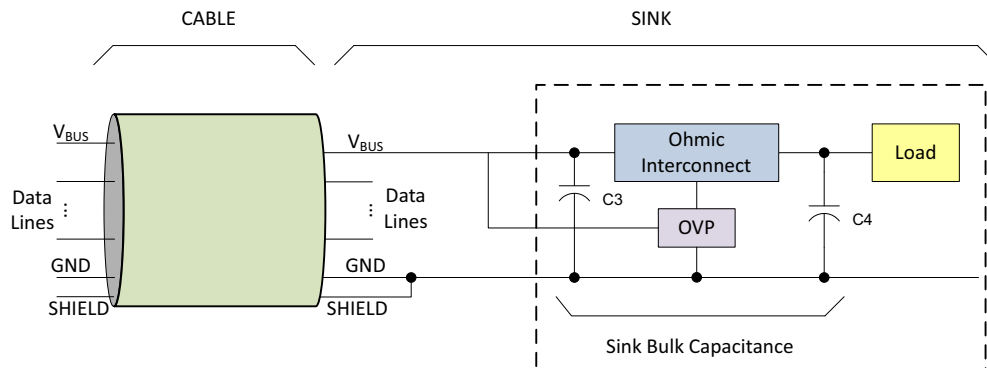
- **Shall Not** draw more than [USB Type-C 2.4] USB Type-C Current from *V_{BUS}* while in a *Default Contract* or *Implicit Contract*.
- **Shall** follow the requirements as specified in [Section 7.1.5, "Response to Hard Resets"](#) when **Hard Reset Signaling** is received.
- **Shall** control *V_{BUS}* in-rush current when increasing current consumption according to [USB 2.0] or [USB 3.2] as appropriate.

7.2.2 Sink Bulk Capacitance

The Sink bulk capacitance consists of C3 and C4 as shown in [Figure 7.21, "Placement of Sink Bulk Capacitance"](#). The Ohmic Interconnect might consist of PCB traces for power distribution or power switching devices. The Ohmic Interconnect is expected to be part of an input Over Voltage Protection (*Sink OVP*) circuit implemented by the Sink as described in [Section 7.2.9.2, "Input Over Voltage Protection"](#) to protect against excessive *V_{BUS}* input voltage. A Sink **Shall** implement *OVP*. The Sink **Shall Not** rely on the Source output voltage limit for its input *OVP*. The capacitance might be a single capacitor, a capacitor bank or distributed capacitance. An upper bound of **cSnkBulkPd** **Shall Not** be exceeded so that the transient charging, or discharging, of the total bulk capacitance on *V_{BUS}* can be accounted for during voltage transitions.

The Sink bulk capacitance that is within the **cSnkBulk** max or **cSnkBulkPd** max limits is allowed to change to support a newly *Negotiated* power level. The capacitance can be changed when the Sink enters *Sink Standby* or during a voltage transition or when the Sink begins to operate at the new power level. Changing the Sink bulk capacitance **Shall Not** cause a transient current on *V_{BUS}* that violates the present *Contract*. During a *Power Role Swap* the Default Sink **Shall** transition to *Swap Standby* before operating as the *New Source*. Any change in bulk capacitance required to complete the *Power Role Swap* **Shall** occur during *Swap Standby*.

Figure 7.21 Placement of Sink Bulk Capacitance



7.2.3 Sink Standby

The Sink **Shall** transition to *Sink Standby* before a positive voltage transition of *V_{BUS}*. During *Sink Standby* the Sink **Shall** reduce the current drawn to **iSnkStdby**. This allows the Source to manage the voltage transition as well as supply sufficient operating current to the Sink to maintain PD operation during the transition. The Sink **Shall** complete this transition to *Sink Standby* within **tSnkStdby** after evaluating the **Accept Message** from the Source. The transition when returning to Sink operation from *Sink Standby* **Shall** be completed within **tSnkNewPower**. The **iSnkStdby** requirement **Shall** only apply if the Sink current draw is higher than this level.

See [Section 7.3, "Transitions"](#) for details.

7.2.3.1 Programmable Power Supply Sink Standby

A *Sink* is not required to transition to *Sink Standby* when operating within the *Negotiated* PPS APDO. A *Sink* **May** consume the Operating Current value in the PPS RDO during PPS output voltage changes. However, prior to operating the *SPR PPS* in *Current Limit*, the *Sink* **Shall** program the PPS Operating Voltage to the lowest practical level that satisfies the *Sink* load requirement. Doing so will minimize the inrush current that occurs when the transition to *Current Limit* occurs. When operating with an *SPR PPS Source* that is in *Current Limit*, the *Sink* **Shall Not** change its load in a manner that exceeds *iPpsCLLoadStepRate* or *iPpsCLLoadReleaseRate*. The load change magnitude **Shall Not** request a change to the *Current Limit* set-point that exceeds *iPpsCLLoadStep*.

If the *Sink* *Negotiates* for a new PPS APDO, that is expected to increase *V_{BUS}* voltage, then the *Sink* **Shall** transition to *Sink Standby* while changing between PPS APDOs as described in [Section 7.3.1, "Transitions caused by a Request Message"](#).

7.2.4 Suspend Power Consumption

When *Source* has set its *USB Suspend Supported* flag (see [Section 6.4.1.2.1.2, "USB Suspend Supported"](#)), a *Sink* **Shall** go to the lowest power state during USB suspend. The lowest power state **Shall** be *pSnkSusp* or lower for a PDUSB Peripheral and *pHubSusp* or lower for a PDUSB Hub. There is no requirement for the *Source* voltage to be changed during USB suspend.

7.2.5 Zero Negotiated Current

When a *Sink* Requests zero current as part of a power *Negotiation* with a *Source*, the *Sink* **Shall** go to the lowest power state, *pSnkSusp* or lower, where it can still communicate using PD signaling.

7.2.6 Transient Load Behavior

When a *Sink*'s operating current changes due to a load step, load release or any other change in load level, the positive or negative overshoot of the new load current **Shall Not** exceed the range defined by *iOvershoot*. For the purposes of measuring *iOvershoot* the new load current value is defined as the average steady state value of the load current after the load step has settled. The rate of change of any shift in *Sink* load current during normal operation **Shall Not** exceed *iLoadStepRate* (for load steps) and *iLoadReleaseRate* (for load releases) as measured at the *Sink* receptacle.

The *Sink*'s operating current **Shall Not** change faster than the value reported in the *Source*'s Load Step Slew Rate in its *Voltage Regulation* bit field and **Shall** ensure that PD Communications meet the transmit and receive masks as specified in [Section 5.8.2, "Transmit and Receive Masks"](#).

7.2.7 Swap Standby for Sinks

The *Sink* functionality in a *Dual-Role Power Port* **Shall** support *Swap Standby*. *Swap Standby* occurs for the *Sink* after evaluating the *Accept* Message from the *Source* during a *Power Role Swap*. While in *Swap Standby* the *Sink*'s current draw **Shall Not** exceed *iSnkSwapStdby* from *V_{BUS}* and the *Dual-Role Power Port* **Shall** be configured as a *Source* after *V_{BUS}* has been discharged to *vSafe0V* by the existing *Initial Source*. The *Sink*'s USB connection **Should Not** be reset even though *vSafe5V* is not present on the *V_{BUS}* conductor (see [Section 9.1.2, "Mapping to USB Device States"](#)). The time for the *Sink* to transition to *Swap Standby* **Shall** be no more than *tSnkSwapStdby*. When in *Swap Standby* the *Sink* has relinquished its *Power Role* as *Sink* and will prepare to become the *New Source*. The transition time from *Swap Standby* to *New Source* **Shall** be no more than *tNewSrc*.

7.2.8 Sink Peak Current Operation

Sinks **Shall** only make use of a *Source* overload capability when the corresponding *Fixed Supply PDO* Peak Current (see [Section 6.4.1.2.1.8, "Peak Current"](#)) or *AVS APDO* Peak Current (see [Section 6.4.1.2.4.3.2, "Peak Current"](#)) bits are set to 01b, 10b and 11b. *Sinks* **Shall** manage thermal aspects of the overload event by not exceeding the average *Negotiated* output of a *Fixed Supply* or *AVS* that supports Peak Current operation.

Sinks that depend on the Peak Current capability for enhanced system performance **Shall** also function correctly when *Attached* to a *Source* that does not offer the Peak Current capability or when the Peak Current capability has been inhibited by the *Source*.

7.2.9 Robust Sink Operation

7.2.9.1 Sink Bulk Capacitance Discharge at Detach

When a *Source* is *Detached* from a *Sink*, the *Sink* **Shall** continue to draw power from its input bulk capacitance until *VBUS* is discharged to **vSafe5V** or lower by no longer than **tSafe5V** from the *Detach* event. This safe *Sink* requirement **Shall** apply to all *Sinks* operating with a *Negotiated VBUS* level greater than **vSafe5V** and **Shall** apply during all low power and high-power operating modes of the *Sink*.

If the *Detach* is detected during a *Sink* low power state, such as USB Suspend, the *Sink* can then draw as much power as needed from its bulk capacitance since a *Source* is no longer *Attached*. In order to achieve a successful *Detach* detect based on *VBUS* voltage level droop, the *Sink* power consumption **Shall** be high enough so that *VBUS* will decay below **vSrcValid**(min) well within **tSafe5V** after the *Source* bulk capacitance is removed due to the *Detach*. Once adequate *VBUS* droop has been achieved, a discharge circuit can be enabled to meet the safe *Sink* requirement.

To illustrate the point, the following set of *Sink* conditions will not meet the safe *Sink* requirement without additional discharge circuitry:

- *Negotiated VBUS* = 20V.
- Maximum allowable supplied *VBUS* voltage = 21.55V.
- Maximum bulk capacitance = 30μF.
- Power consumption at *Detach* = 12.5mW.

When the *Detach* occurs (hence removal of the *Source* bulk capacitance) the 12.5mW power consumption will draw down the *VBUS* voltage from the worst-case maximum level of 21.55V to 17V in approximately 205ms. At this point, with *VBUS* well below **vSrcValid** (min) an approximate 100mW discharge circuit can be enabled to increase the rate of *Sink* bulk capacitance discharge and meet the safe *Sink* requirement. The power level of the discharge circuit is dependent on how much time is left to discharge the remaining voltage on the *Sink* bulk capacitance. If a *Sink* has the ability to detect the *Detach* in a different manner and in much less time than **tSafe5V**, then this different manner of detection can be used to enable a discharge circuit, allowing even lower power dissipation during low power modes such as USB Suspend.

In most applications, the safe *Sink* requirement will limit the maximum *Sink* bulk capacitance well below the **cSnkBulkPd** limit. A *Detach* occurring during *Sink* high power operating modes must quickly discharge the *Sink* bulk capacitance to **vSafe5V** or lower as long as the *Sink* continues to draw adequate power until *VBUS* has decayed to **vSafe5V** or lower.

7.2.9.2 Input Over Voltage Protection

Sinks **Shall** implement input Over-Voltage Protection (OVP) to prevent damage from input voltage that exceeds the voltage handling capability of the *Sink*. The definition of voltage handling capability is left to the discretion of the *Sink* implementation. The over voltage response of *Sinks* **Shall Not** interfere with normal PD operation and **Shall** account for **vSrcNew**, **vSrcValid** or **vPpsNew**, **vPpsValid** as determined by the *Negotiated VBUS* value. SPR *Sinks* **Should** tolerate input voltages as high as **vSprMax** and **Shall** meet applicable safety requirements if **vSprMax** is exceeded. Likewise, EPR *Sinks* **Should** tolerate input voltages as high as **vEprMax** and **Shall** meet applicable safety requirements if **vEprMax** is exceeded.

Sinks **Should** attempt to send **Hard Reset** Signaling when OVP engages followed by an **Alert** Message indicating an OVP event once an *Explicit Contract* has been established. The OVP response **May** engage at either the *Port* or system level. Systems or ports that have engaged OVP **Shall** resume *USB Default Operation* when the *Source* has re-established **vSafe5V** on *VBUS*.

The *Sink* **Shall** be able to *Re-negotiate* with the *Source* after resuming *USB Default Operation*. The decision of how to respond to *Re-negotiation* after an OVP event is left to the discretion of the *Sink* implementation.

The *Sink* **Shall** prevent continual system or *Port* cycling if OVP continues to engage after initially resuming either *USB Default Operation* or *Re-negotiation*. Latching off the *Port* or system is an acceptable response to recurring over voltage.

7.2.9.3 Over Temperature Protection

Sinks **Shall** implement over temperature protection (*OTP*) to prevent damage from temperature that exceeds the thermal capability of the Sink. The definition of thermal capability and the monitoring locations used to trigger the over temperature protection are left to the discretion of the Sink implementation.

Sinks **Shall** attempt to send **Hard Reset Signaling** when over temperature protection engages followed by an **Alert Message** indicating an *OTP* event once an *Explicit Contract* has been established. The over temperature protection response **May** engage at either the Port or system level. Systems or ports that have engaged over temperature protection **Should** attempt to resume *USB Default Operation* after sufficient cooling is achieved and **May** latch off to protect the Port or system. The definition of sufficient cooling is left to the discretion of the Sink implementation.

The Sink **Shall** be able to *Re-negotiate* with the Source after resuming *USB Default Operation*. The decision of how to respond to *Re-negotiation* after an over temperature event is left to the discretion of the Sink implementation.

The Sink **Shall** prevent continual system or Port cycling if over temperature protection continues to engage after initially resuming either *USB Default Operation* or *Re-negotiation*. Latching off the Port or system is an acceptable response to recurring over temperature.

7.2.9.4 Over Current Protection

Sinks that operate with a Programmable Power Supply **Shall** implement their own internal current protection mechanism to protect against internal *V_{BUS}* current faults as well as erratic Source current regulation. The Sink **Shall** never draw higher current than the Maximum Current value in the PPS *APDO*.

7.2.10 Fast Role Swap

As described in [Section 7.1.13, "Fast Role Swap"](#) a *Fast Role Swap* limits the interruption of *V_{BUS}* power to a bus powered accessory connected to a Hub DFP that has a *UFP Attached* to a power source and a *DRP Attached* to a Host Port that supports *DRP*. This configuration is shown in [Figure 7.16, "V_{BUS} Power during Fast Role Swap"](#).

The Host *DRP*, upon establishing an *Explicit Contract*, **Shall** query the Initial Source's Sink Capabilities to determine whether the Initial Source supports *Fast Role Swap*, and what level of current it requires. If the **Sink Capabilities Message** received from the Initial Source has at least one of the **Fast Role Swap required USB Type-C Current** bits set, and the Host *DRP* is able to source the requested current at 5V, the Host *DRP* **May** arm itself for *Fast Role Swap*. If the Host *DRP* has not queried the Sink Capabilities from the Initial Source, or if the **Sink Capabilities Message** reports no *Fast Role Swap* support or a current that is beyond what the Host *DRP* is able or willing to source in the event of a *Fast Role Swap*, the Host *DRP* **Shall Not** arm itself for *Fast Role Swap* and **Shall Ignore** any *Fast Role Swap Requests* that are detected.

When the Host *DRP* that supports *Fast Role Swap* detects the *FFast Role Swap Request*, the Host *DRP* **Shall** stop sinking current and **Shall** be ready and able to source **vSafe5V** if the residual *V_{BUS}* voltage level at the Host *DRP* connector is greater than **vSafe5V**. When the residual *V_{BUS}* voltage level at the Host *DRP* connector discharges below **vSafe5V**(min) the Host *DRP* as the New Source **Shall** supply **vSafe5V** to the Hub *DRP* within **tSrcFRSwap**. The Host *DRP* **Shall Not** enable *V_{BUS}* discharge circuitry when changing Power Roles from Initial Sink to New Source.

The New Source **Shall** supply **vSafe5V** at USB Type-C Current (see [\[USB Type-C 2.4\]](#)) at the value Advertised in the **Fast Role Swap required USB Type-C Current** field (see [Section 6.4.1.3.1.6, "Fast Role Swap USB Type-C Current"](#)). All Source requirements **Shall** apply to the New Source after the *Fast Role Swap* is complete. The *Fast Role Swap* response of the Hub *DRP* is described in [Section 7.1.13, "Fast Role Swap"](#) since the Hub *DRP* is operating as the Initial Source prior to the *Fast Role Swap*.

After the Host *DRP* is providing *V_{BUS}* power to the Hub *DRP*, a **PS_RDY Message** **Shall** be sent to the Hub *DRP* as defined by the *Fast Role Swap Request* and the AMS detailed in [Section 7.3.4, "Transitions Caused by Fast Role Swap"](#).

7.3 Transitions

The following sections illustrate the power supply's response to various types of *Negotiations*. The *Negotiations* are triggered by certain *Messages* or *Signaling*. It provides examples of the transitions and is organized around each of the *Messages* and *Signals* that result in a response from the power supply. The response to a *Message* or *Signal* can result in different transitions depending upon the power supply's starting conditions and the requested change.

- Transitions caused by a *Request Message*:
 - Generic transition between *(A)PDO* s:
 - Increase the current.
 - Increase the voltage.
 - Increase the voltage and the current.
 - Increase the voltage and decrease the current.
 - Decrease the voltage and increase the current.
 - Decrease the voltage and the current.
 - No change in Current or voltage.
 - Transitions within the same *PDO* (*Fixed Supply*, *Battery Supply*, *Variable Supply*):
 - Increase the current.
 - Decrease the current.
 - No change in current.
 - Transitions within the same PPS *APDO*:
 - Increasing the Programmable Power Supply (PPS) voltage.
 - Decreasing the Programmable Power Supply (PPS) voltage.
 - Increasing the Programmable Power Supply (PPS) Current.
 - Decreasing the Programmable Power Supply (PPS) Current.
 - Same Request Programmable Power Supply (PPS).
 - Transitions within the same *AVS APDO*:
 - Increasing the *Adjustable Voltage Supply* (AVS) voltage
 - Decreasing the *Adjustable Voltage Supply* (AVS) voltage
 - Same Request *Adjustable Voltage Supply* (AVS)
- Transitions caused by the *PR_Swap Message*:
 - *Source* requests a *Power Role Swap*
 - *Sink* requests a *Power Role Swap*
- Transitions caused by *Hard Reset Signaling*:
 - *Source* issues *Hard Reset Signaling*.
 - *Sink* issues *Hard Reset Signaling*.
- Transitions caused by the *Fast Role Swap Request*:
 - *Source* asserts R_d at its preferred *[USB Type-C 2.4]* current.

7.3.1 Transitions caused by a Request Message

This section describes transitions that are caused by a *Request Message*.

7.3.1.1 Changing the Source between Different (A)PDOs

In these transition descriptions the term (A)PDO is used to describe any *Power Data Object*, regardless of whether it is a *PDO* or an *APDO* in the *Capabilities Message*.

This section describes transitions in response to a *Request Message*:

- From one (A)PDO to another (A)PDO
- From an *Implicit Contract* to an *Explicit Contract*
- From [\[USB Type-C 2.4\]](#)operation to the *First Explicit Contract*

These transitions usually result in a voltage change but is not required.

The interaction of the *Device Policy Manager*, the *Port Policy Engine* and the Power Supply that **shall** be followed when increasing the current is shown in [Figure 7.23, "Transition Diagram for Increasing the Voltage and Current"](#) and [Figure 7.25, "Transition Diagram for Decreasing the Voltage and Increasing the Current"](#).

The *Source* voltage as the transition starts **shall** be any voltage within the **Valid** *V_{BUS}* range of the previous *Source PDO* or *APDO*. The *Source* voltage after the transition is complete **shall** be any voltage within the **Valid** *V_{BUS}* range of the *New Source PDO* or *APDO*. The sequence that **shall** be followed is described in [Table 7.2, "Sequence Diagram for Increasing the Voltage and Current"](#) and [Table 7.4, "Sequence Description for Decreasing the Voltage and Increasing the Current"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In this figure, the *Sink* has previously sent a *Request Message* to the *Source*.

The voltage is considered to increase if the change from V_{OLD} to V_{NEW} is greater than **vSmallStep**. The determination **shall** be based on the nominal (A)PDO voltage before and after, unless either (A)PDO is *Battery Supply* or *Variable Supply* when the worst case of the following is assumed in making this determination.

- Minimum voltage to voltage.
- Minimum voltage to Maximum voltage.
- Voltage to Maximum voltage.

The following sections begin with a description of the generic process followed by more specific examples of the most common transitions.

7.3.1.1.1 Examples of changes from one (A)PDO to another (A)PDO

The seven examples of (A)PDO change transitions below illustrate the most common transitions.

7.3.1.1.1.1 Increasing the Voltage

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed when changing from one (A)PDO to another while increasing the voltage is shown in [Figure 7.22, "Transition Diagram for Increasing the Voltage"](#). The sequence that **shall** be followed is described in [Table 7.1, "Sequence Description for Increasing the Voltage"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.22, "Transition Diagram for Increasing the Voltage"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.22 Transition Diagram for Increasing the Voltage

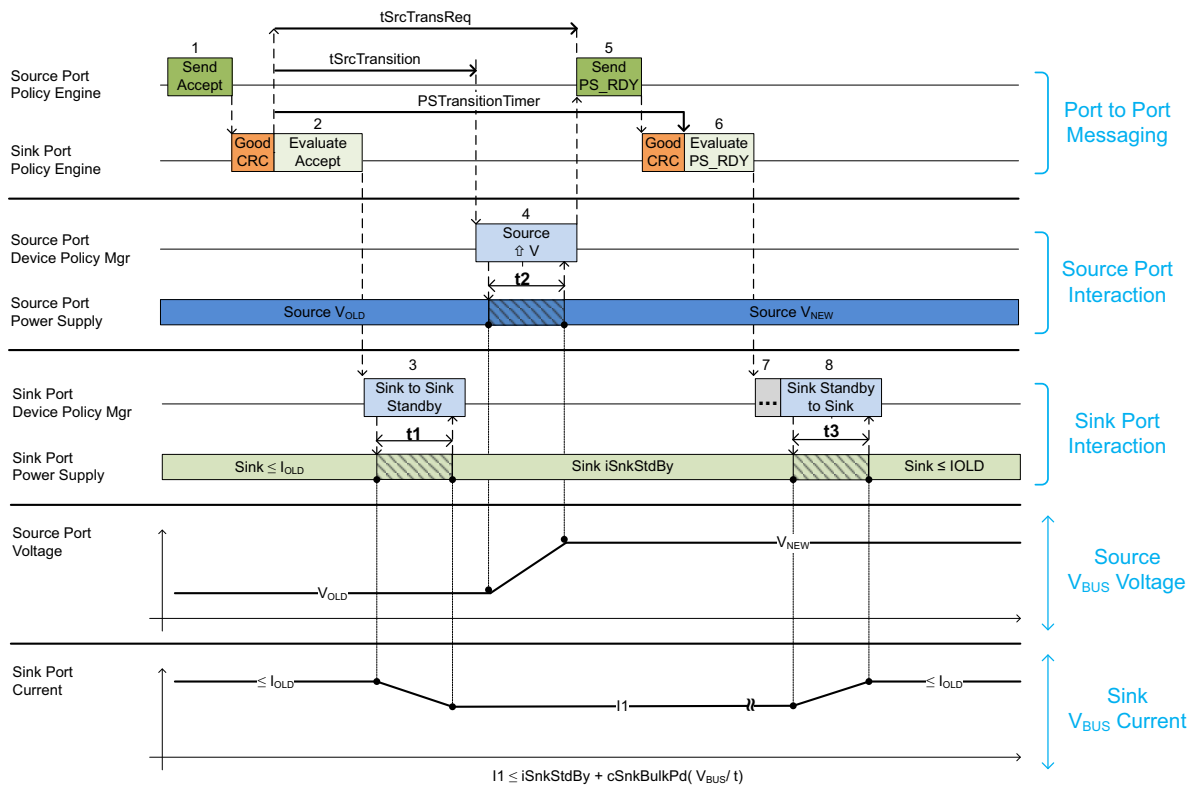


Table 7.1 Sequence Description for Increasing the Voltage

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3		<i>Policy Engine tells the Device Policy Manager to instruct the power supply to reduce current drawn to iSnkStdby within tSnkStdby (t1); t1 Shall complete before tSrcTransition. The Sink Shall Not violate transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.</i>
4	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t2). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
5	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
6	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
7		<i>The Sink May begin operating at the new power level any time after evaluation of the PS_RDY Message. This time duration is indeterminate.</i>
8		<i>The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. The time duration (t3) depends on the magnitude of the load change.</i>

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed when changing from one (A)PDO to another while increasing the voltage and current is shown in [Figure 7.23, "Transition Diagram for Increasing the Voltage and Current"](#). The sequence that **shall** be followed is described in [Table 7.2, "Sequence Diagram for Increasing the Voltage and Current"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.23, "Transition Diagram for Increasing the Voltage and Current"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.23 Transition Diagram for Increasing the Voltage and Current

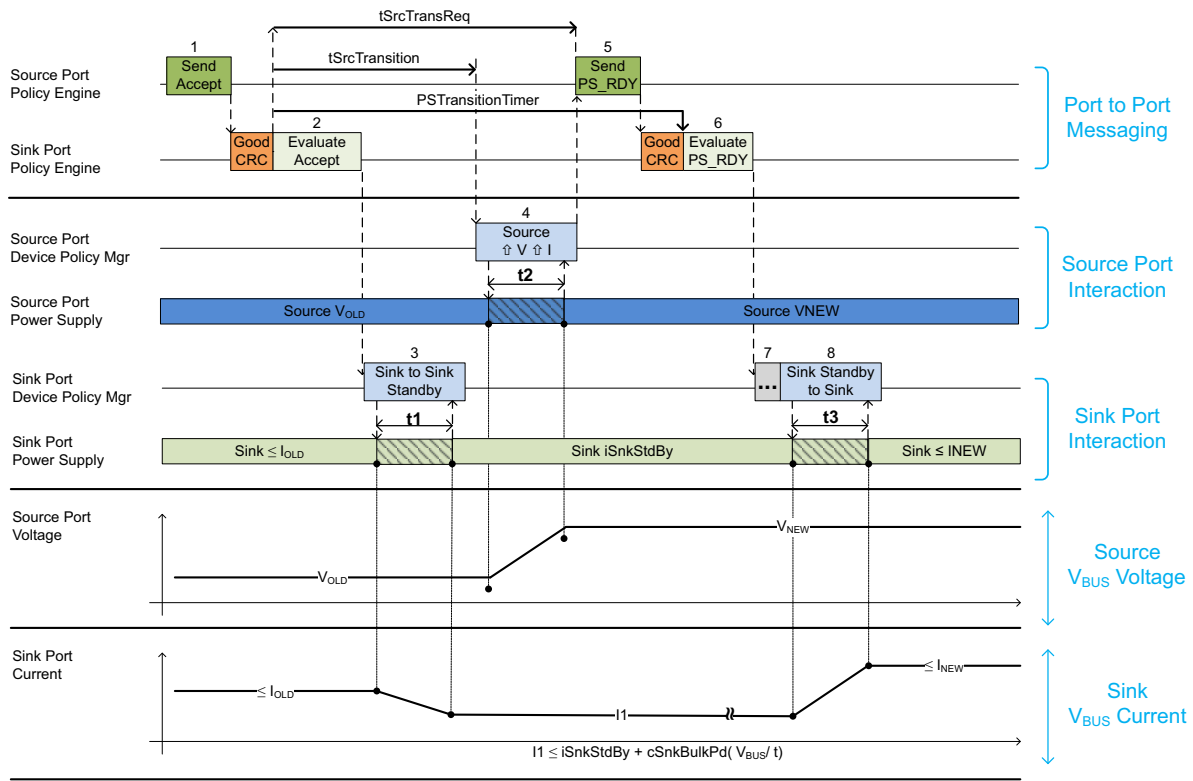


Table 7.2 Sequence Diagram for Increasing the Voltage and Current

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3		<i>Policy Engine tells the Device Policy Manager to instruct the power supply to reduce current drawn to iSnkStdby within tSnkStdby (t1); t1 Shall complete before tSrcTransition. The Sink Shall Not violate transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.</i>
4	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t2). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
5	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
6	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out, the Sink sends Hard Reset Signaling.</i>
7		<i>The Sink May begin operating at the new power level any time after evaluation of the PS_RDY Message. This time duration is indeterminate.</i>
8		<i>The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. The time duration (t3) depends on the magnitude of the load change.</i>

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when changing from one (A)PDO to another while increasing the voltage and decreasing the current is shown in [Figure 7.24, "Transition Diagram for Increasing the Voltage and Decreasing the Current"](#). The sequence that **Shall** be followed is described in [Table 7.3, "Sequence Description for Increasing the Voltage and Decreasing the Current"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.24, "Transition Diagram for Increasing the Voltage and Decreasing the Current"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.24 Transition Diagram for Increasing the Voltage and Decreasing the Current

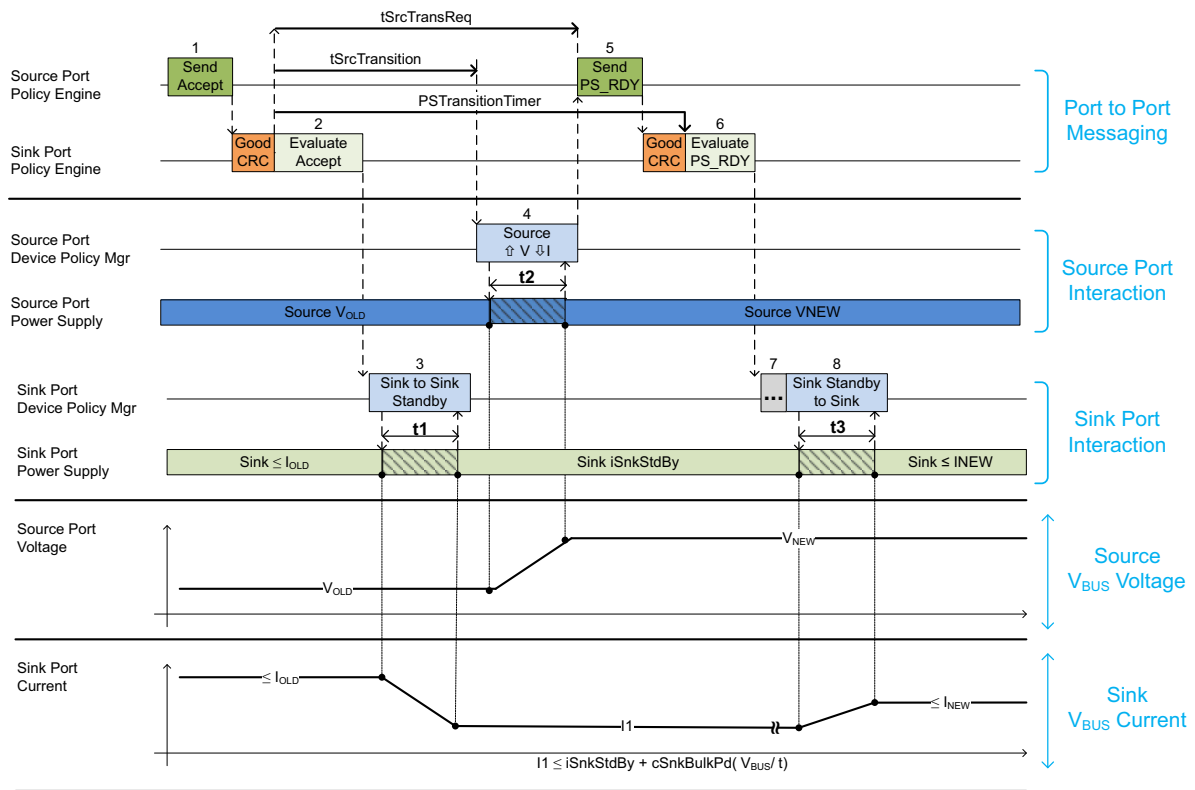


Table 7.3 Sequence Description for Increasing the Voltage and Decreasing the Current

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3		<i>Policy Engine tells the Device Policy Manager to instruct the power supply to reduce current drawn to iSnkStdby within tSnkStdby (t1); t1 Shall complete before tSrcTransition. The Sink Shall Not violate transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.</i>
4	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t2). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
5	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
6	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
7		<i>The Sink May begin operating at the new power level any time after evaluation of the PS_RDY Message. This time duration is indeterminate.</i>
8		<i>The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. The time duration (t3) depends on the magnitude of the load change.</i>

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when changing from one (A)PDO to another while decreasing the voltage and increasing the current is shown in [Figure 7.25, "Transition Diagram for Decreasing the Voltage and Increasing the Current"](#). The sequence that **Shall** be followed is described in [Table 7.4, "Sequence Description for Decreasing the Voltage and Increasing the Current"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.25, "Transition Diagram for Decreasing the Voltage and Increasing the Current"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.25 Transition Diagram for Decreasing the Voltage and Increasing the Current

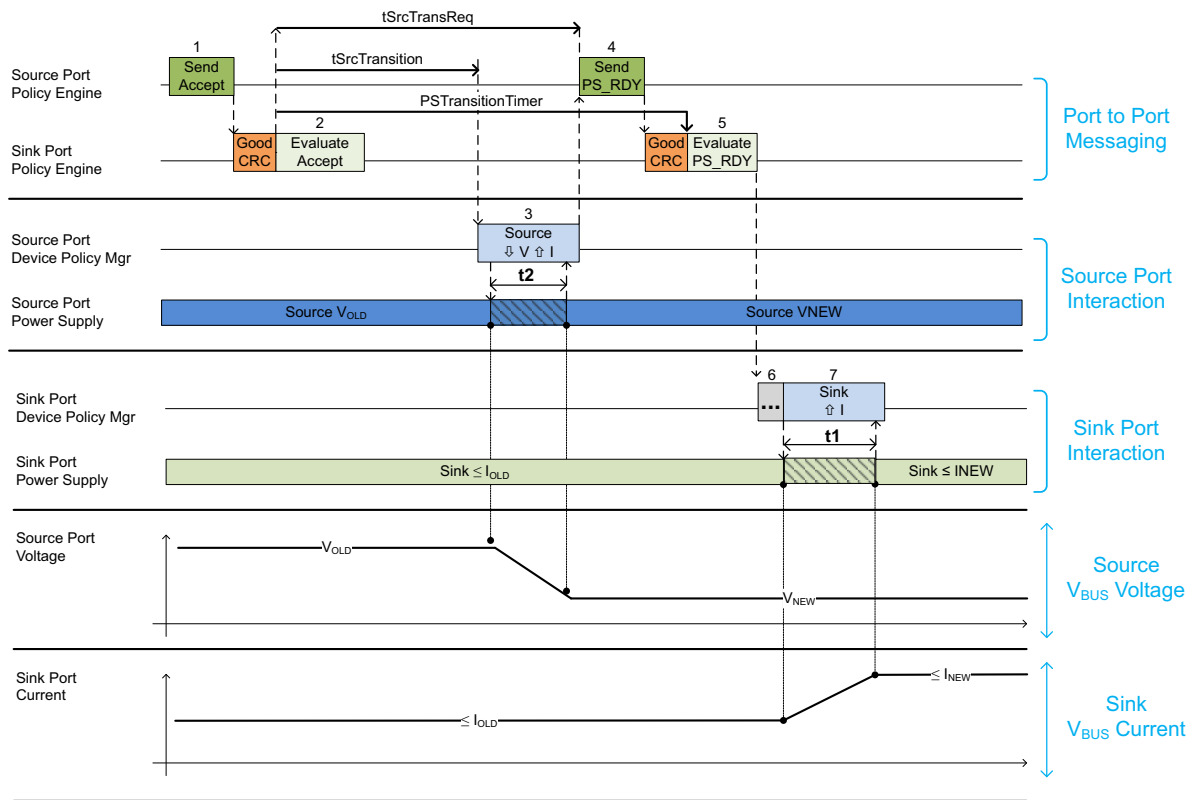


Table 7.4 Sequence Description for Decreasing the Voltage and Increasing the Current

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t1). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then evaluates the PS_RDY Message from the Source and tells the Device Policy Manager it is okay to operate at the new power level.</i>
6		<i>The Sink May begin operating at the new power level any time after evaluation of the PS_RDY Message. This time duration is indeterminate.</i>

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed when changing from one (A)PDO to another while decreasing the voltage is shown in [Figure 7.26, "Transition Diagram for Decreasing the Voltage"](#). The sequence that **shall** be followed is described in [Table 7.5, "Sequence Description for Decreasing the Voltage"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.26, "Transition Diagram for Decreasing the Voltage"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.26 Transition Diagram for Decreasing the Voltage

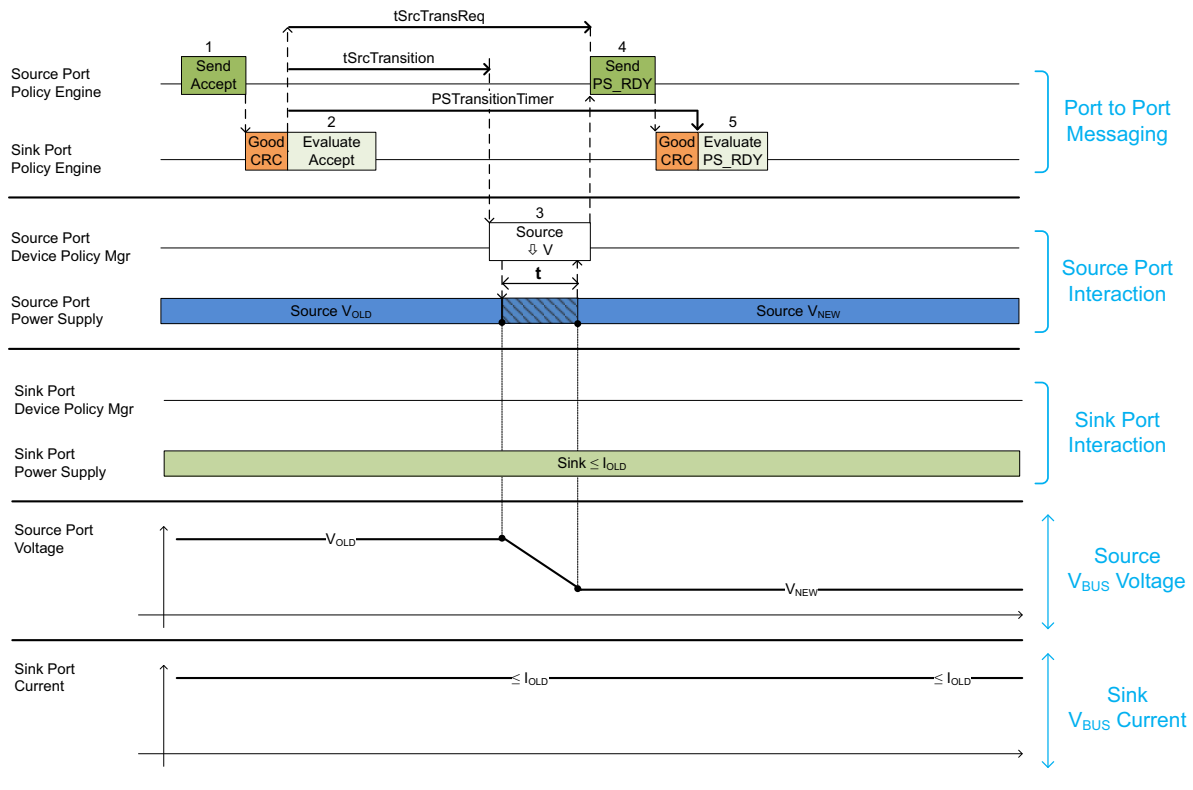


Table 7.5 Sequence Description for Decreasing the Voltage

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t1). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when changing from one (A)PDO to another while decreasing the voltage and current is shown in [Figure 7.28, "Transition Diagram for no change in Current or Voltage"](#). The sequence that **Shall** be followed is described in [Table 7.6, "Sequence Description for Decreasing the Voltage and the Current"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.27, "Transition Diagram for Decreasing the Voltage and the Current"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.27 Transition Diagram for Decreasing the Voltage and the Current

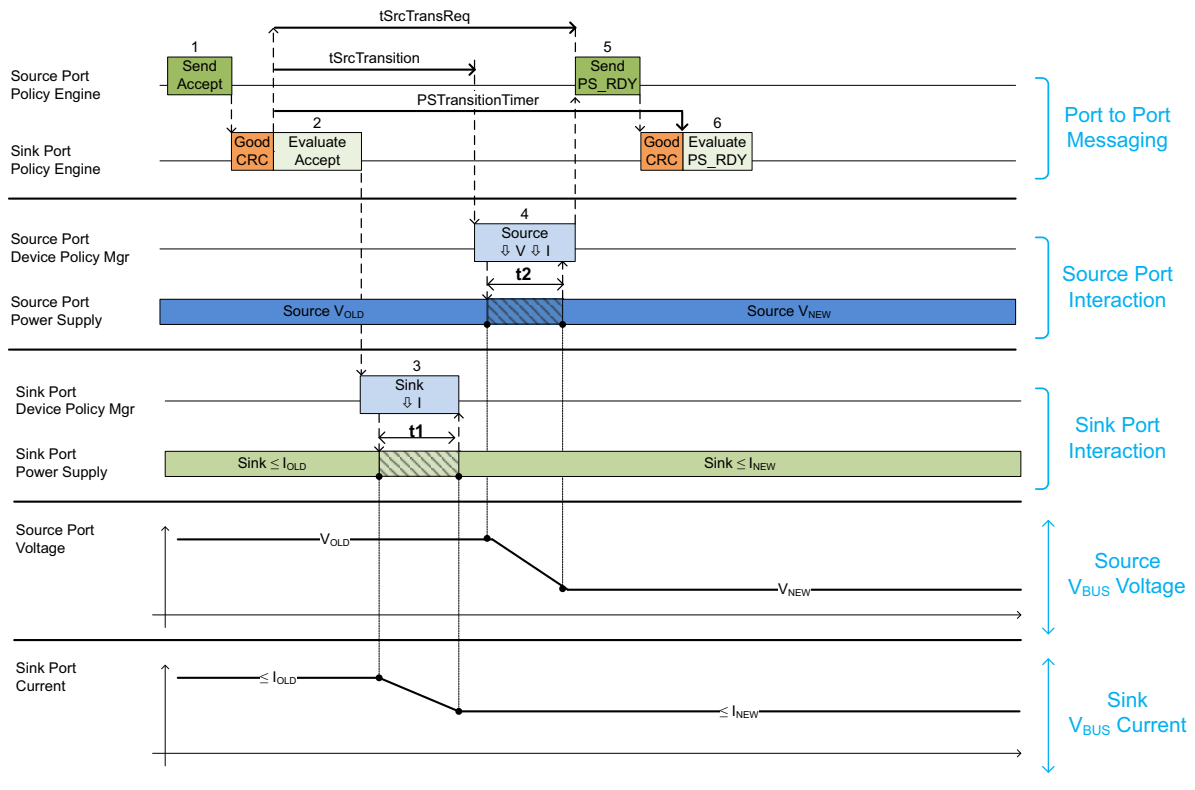


Table 7.6 Sequence Description for Decreasing the Voltage and the Current

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3		<i>The Sink Shall be able to operate with lower current within tSnkNewPower (t1); t1 Shall complete before tSrcTransition. The Sink Shall Not violate transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.</i>
4	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t2). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
5	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
6	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
7		<i>The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.</i>

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when changing from one (A)PDO to another while the *Sink* requests the same voltage and Current as it is currently operating at is shown in [Figure 7.28, "Transition Diagram for no change in Current or Voltage"](#). The sequence that **Shall** be followed is described in [Table 7.7, "Sequence Description for no change in Current or Voltage"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.28, "Transition Diagram for no change in Current or Voltage"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.28 Transition Diagram for no change in Current or Voltage

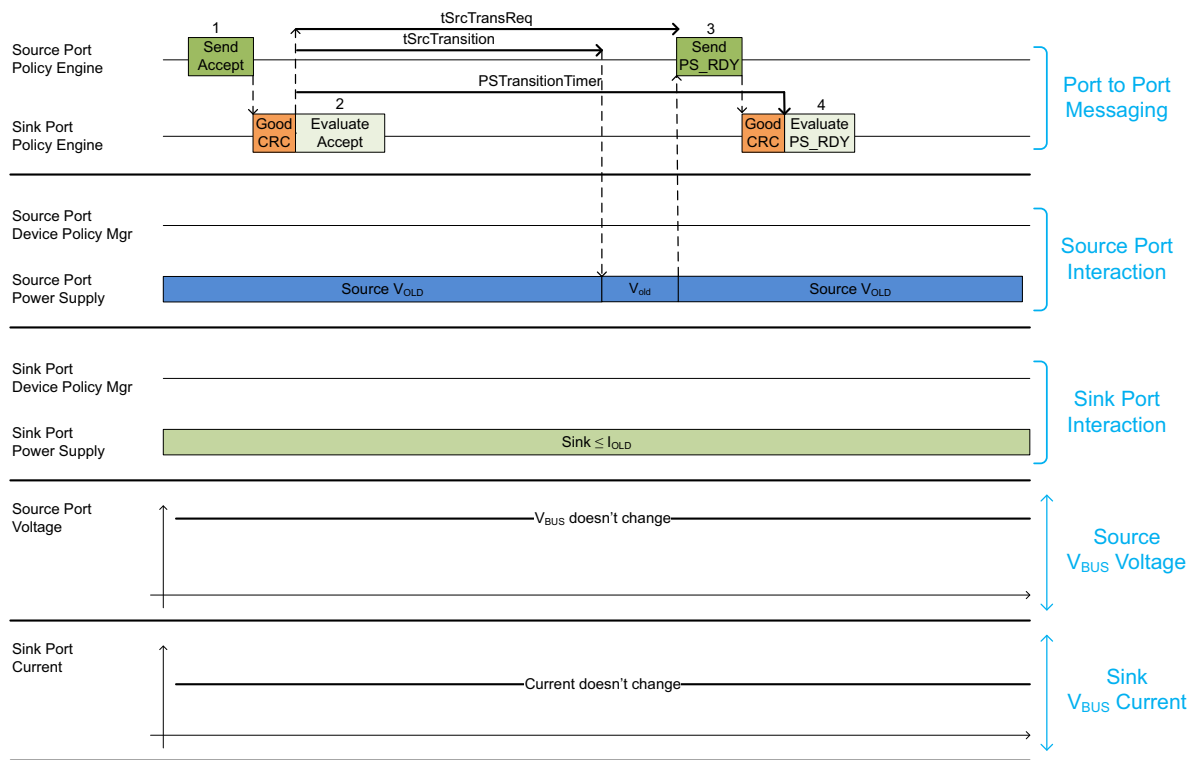


Table 7.7 Sequence Description for no change in Current or Voltage

Step	Source Port	Sink Port
1	Policy Engine sends the Accept Message to the Sink.	Policy Engine receives the Accept Message.
2	Protocol Layer receives the GoodCRC Message from the Sink.	Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PStTransitionTimer and evaluates the Accept Message.
3	The Policy Engine waits tSrcTransition then sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.	Policy Engine receives the PS_RDY Message.
4	Policy Engine receives the GoodCRC Message from the Sink. Note: The decision that no power transition is required could be made either by the Device Policy Manager or the power supply depending on implementation.	Protocol Layer sends the GoodCRC Message to the Source. Policy Engine evaluates the PS_RDY Message.

7.3.1.2 Transitions within the same Fixed, Battery or Variable PDO or between Different (A)PDOs

7.3.1.2.1 Increasing the Current Only

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed when increasing the current without changing the voltage is shown in [Figure 7.29, "Transition Diagram for Increasing the Current"](#). The sequence that **shall** be followed is described in [Table 7.8, "Sequence Description for Increasing the Current"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.29, "Transition Diagram for Increasing the Current"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.29 Transition Diagram for Increasing the Current

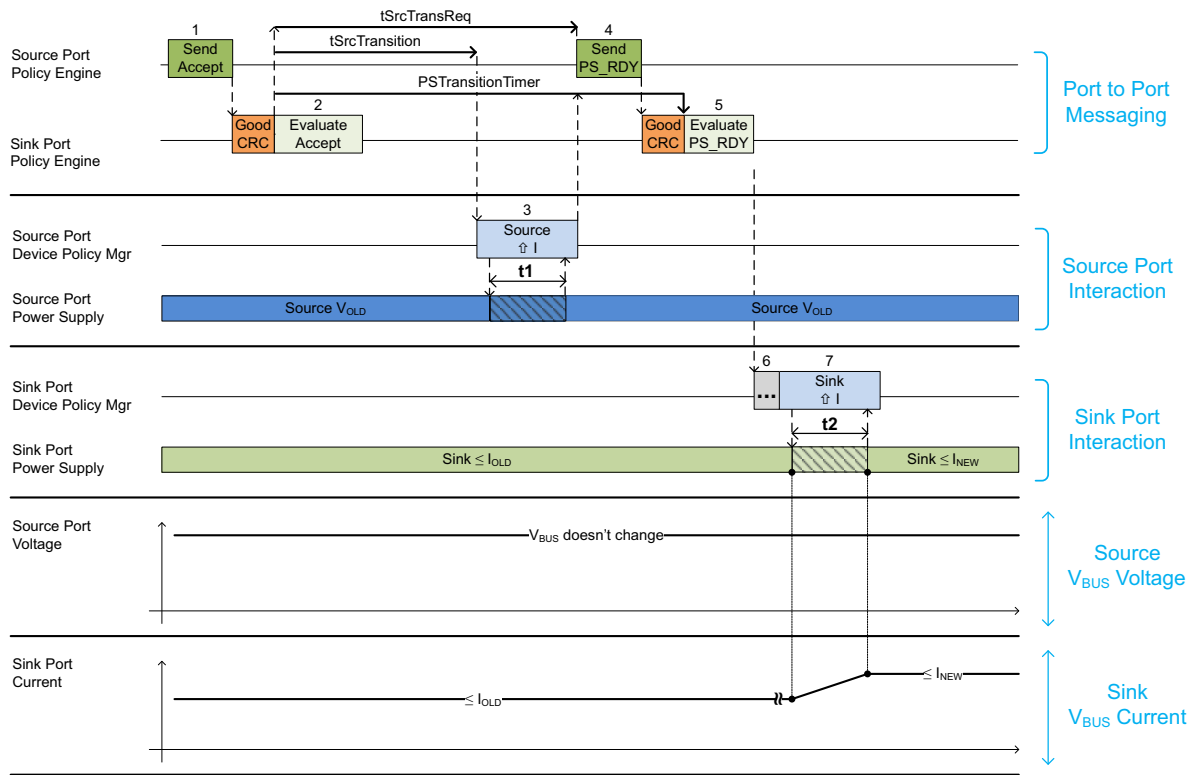


Table 7.8 Sequence Description for Increasing the Current

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t1). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
6		<i>The Sink May begin operating at the new power level any time after evaluation of the PS_RDY Message. This time duration is indeterminate.</i>
7		<i>The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. The time duration (t2) depends on the magnitude of the load change.</i>

7.3.1.2.2 Decreasing the Current Only

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when decreasing the current without changing the voltage is shown in [Figure 7.30, "Transition Diagram for Decreasing the Current"](#). The sequence that **Shall** be followed is described in [Table 7.9, "Sequence Description for Decreasing the Current"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.30, "Transition Diagram for Decreasing the Current"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.30 Transition Diagram for Decreasing the Current

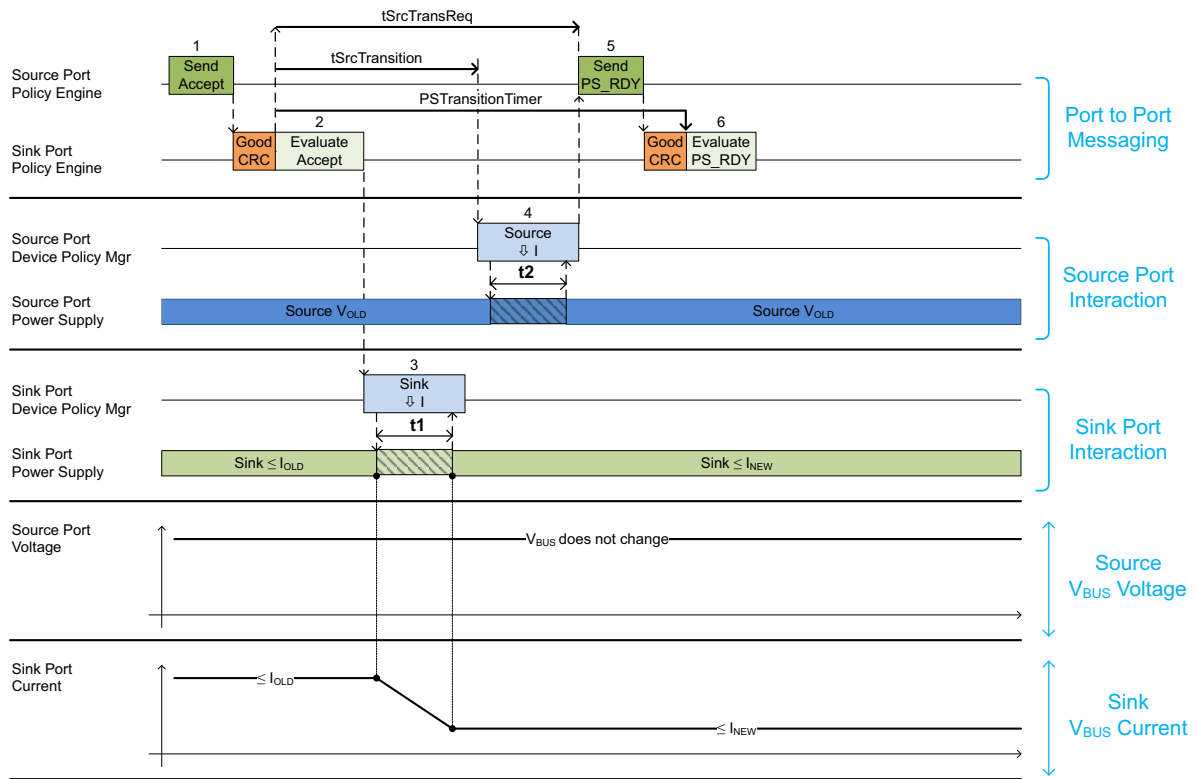


Table 7.9 Sequence Description for Decreasing the Current

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message. Policy Engine tells the Device Policy Manager to instruct the power supply to reduce power consumption.</i>
3		<i>The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. The Sink Shall be able to operate with lower current within tSnkNewPower (t1); t1 Shall complete before tSrcTransition.</i>
4	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability. The power supply Shall be ready to operate at the new power level within tSrcReady (t2). The power supply informs the Device Policy Manager that it is ready to operate at the new power level. The power supply status is passed to the Policy Engine.</i>	
5	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tSrcTransReq of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
6	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new (A)PDO. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>

7.3.1.3 Changing Voltage or Current within the same PPS APDO

7.3.1.3.1 Increasing the Programmable Power Supply (PPS) Voltage

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when increasing the voltage is shown in [Figure 7.31, "Transition Diagram for Increasing the Programmable Power Supply Voltage"](#). The sequence that **Shall** be followed is described in [Table 7.10, "Sequence Description for Increasing the Programmable Power Supply Voltage"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.31, "Transition Diagram for Increasing the Programmable Power Supply Voltage"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.31 Transition Diagram for Increasing the Programmable Power Supply Voltage

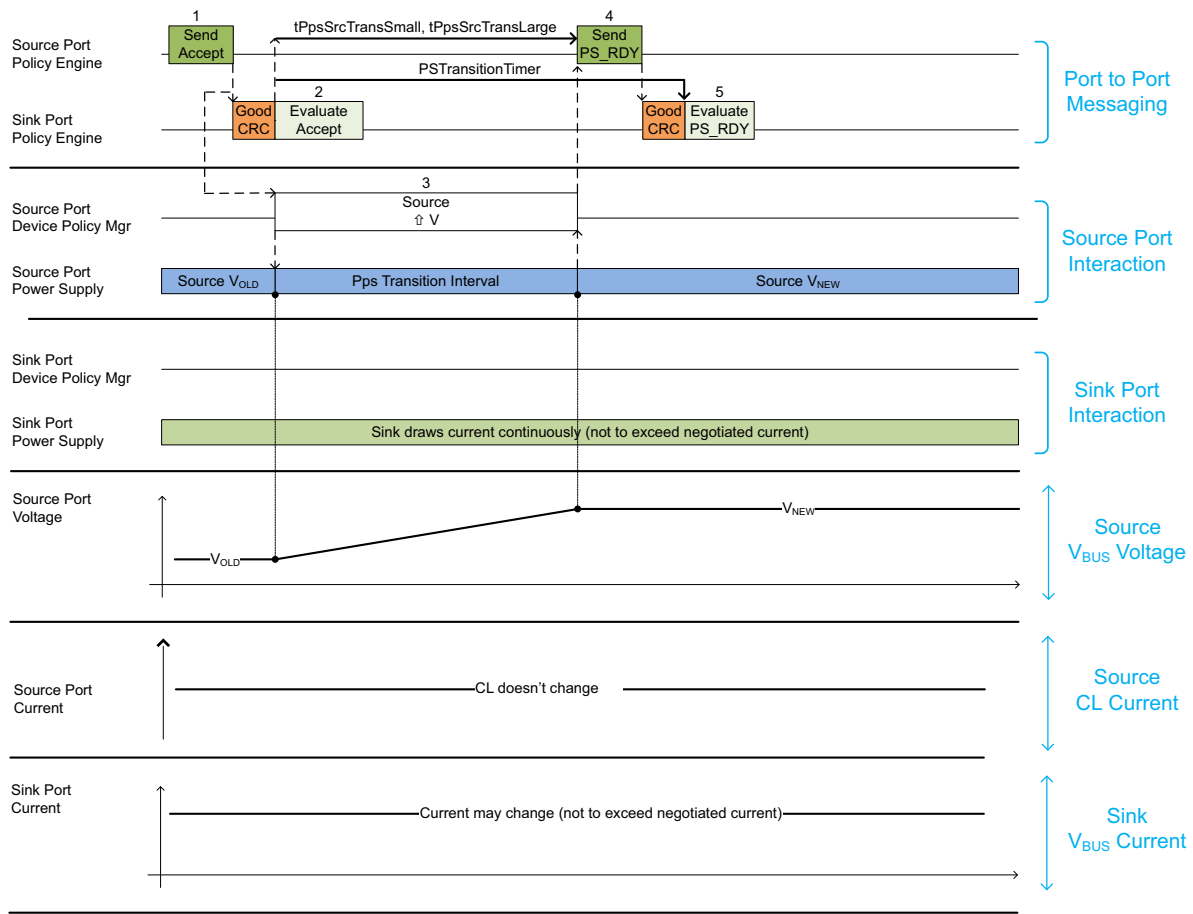


Table 7.10 Sequence Description for Increasing the Programmable Power Supply Voltage

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to increase its output voltage.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>After sending the Accept Message, the Programmable Power Supply starts to increase its output voltage. The Programmable Power Supply new voltage set-point shall be reached by tPpsSrcTransLarge for steps larger than vPpsSmallStep or else by tPpsSrcTransSmall. The power supply informs the Device Policy Manager that it has reached the new set-point and whether V_{BUS} is at the corresponding new level, or if the supply is operating in CL mode. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tPpsSrcTransSmall or tPpsSrcTransLarge of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new voltage set point (corresponding to vPpsNew). If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>

7.3.1.3.2 Decreasing the Programmable Power Supply (PPS) Voltage

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when decreasing the voltage is shown in [Figure 7.32, "Transition Diagram for Decreasing the Programmable Power Supply Voltage"](#). The sequence that **Shall** be followed is described in [Table 7.11, "Sequence Description for Decreasing the Programmable Power Supply Voltage"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.32, "Transition Diagram for Decreasing the Programmable Power Supply Voltage"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.32 Transition Diagram for Decreasing the Programmable Power Supply Voltage

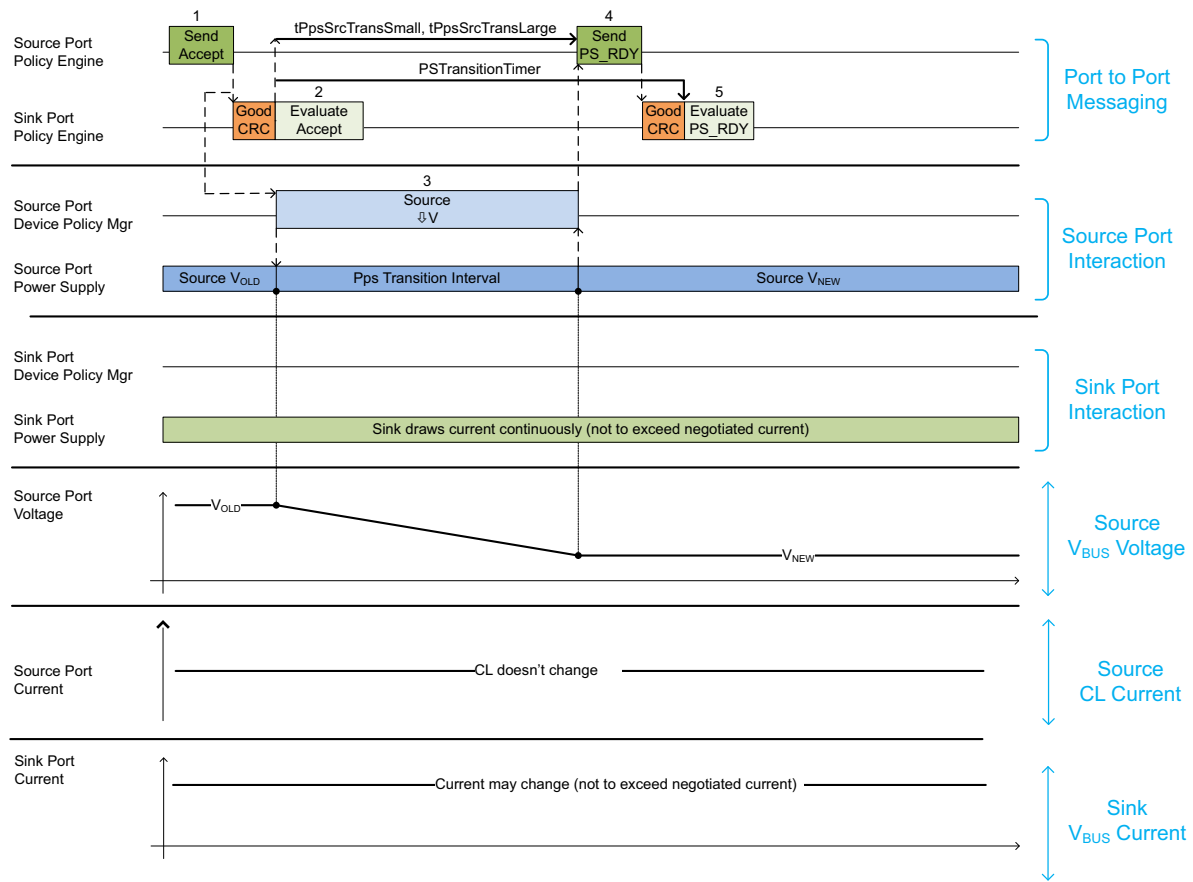


Table 7.11 Sequence Description for Decreasing the Programmable Power Supply Voltage

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to decrease its output voltage.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>After sending the Accept Message, the Programmable Power Supply starts to decrease its output voltage. The Programmable Power Supply new voltage set-point (corresponding to vPpsNew) Shall be reached by tPpsSrcTransLarge for steps larger than vPpsSmallStep or else by tPpsSrcTransSmall. The power supply informs the Device Policy Manager that it has reached the new level. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tPpsSrcTransSmall or tPpsSrcTransLarge of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new voltage set point (corresponding to vPpsNew). If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>

7.3.1.3.3 Increasing the Programmable Power Supply (PPS) Current

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when increasing the current limit in the same *APDO*, not exceeding the maximum for that *APDO* and without changing the requested voltage is shown in [Figure 7.33, "Transition Diagram for increasing the Current in PPS mode"](#). The sequence that **Shall** be followed is described in [Table 7.12, "Sequence Description for increasing the Current in PPS mode"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.33, "Transition Diagram for increasing the Current in PPS mode"](#), the *Sink* has previously sent a **Request** Message to the *Source*.

The *Sink* **May** draw current equal to the increasing *Current Limit* of the *Source* before it has received the **PS_RDY** Message for the new *Request*.

Figure 7.33 Transition Diagram for increasing the Current in PPS mode

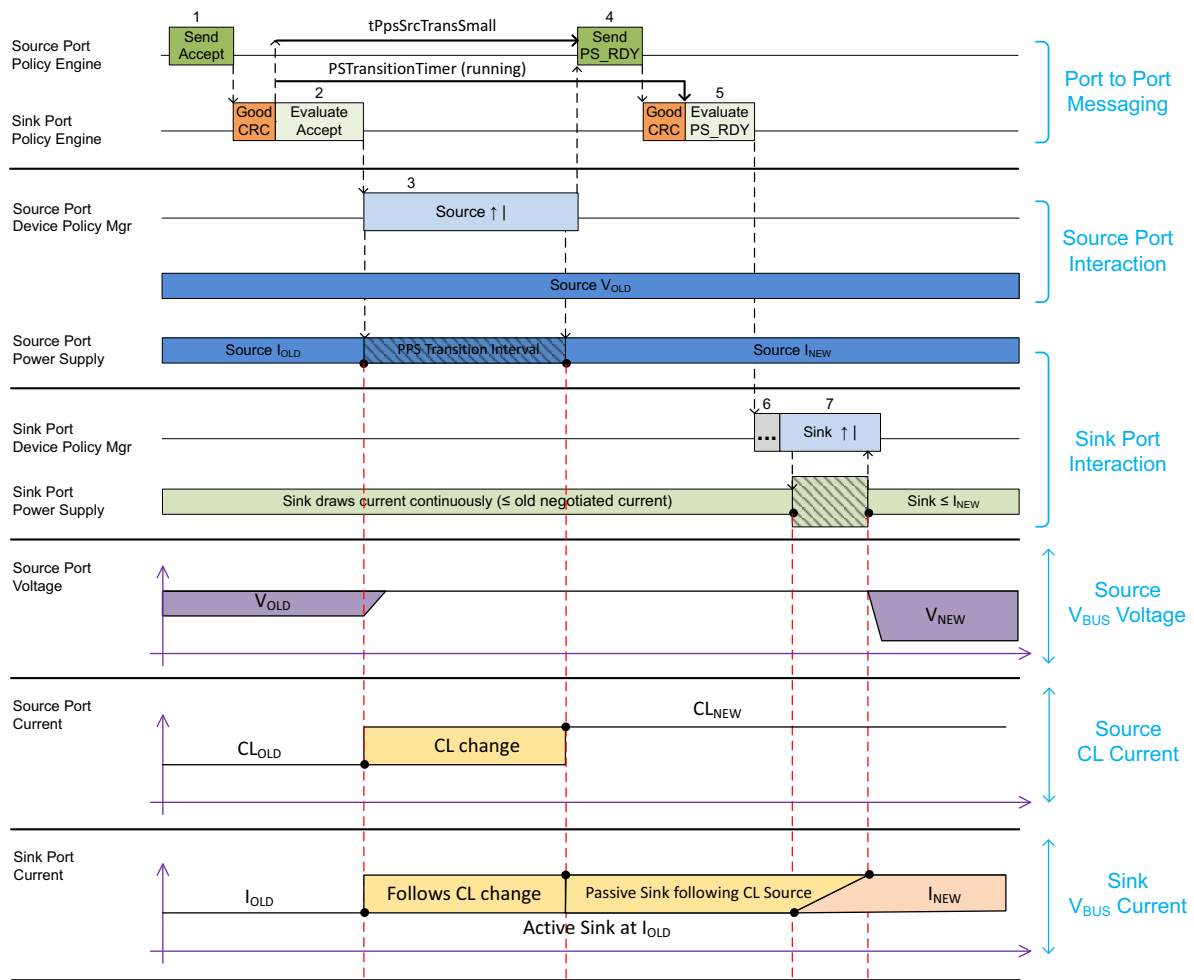


Table 7.12 Sequence Description for increasing the Current in PPS mode

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to increase its set-point for the current limit.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>The Power Supply increases its Current Limit set-point to the new requested value.</i>	<i>The Sink draws current according to the increased Current Limit of the Source.</i>
4	<i>The Policy Engine waits tPpsSrcTransSmall then sends the PS_RDY Message to the Sink starting within tPpsCLProgramSettle of the end of the GoodCRC Message following the Accept Message.</i>	<i>Policy Engine receives the PS_RDY Message.</i>
5	<i>Policy Engine receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source.</i>
6		<i>Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message and tells the Device Policy Manager it can increase the current up to the requested value without the Source going into CL mode.</i> <i>If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
7		<i>The Sink increases its current.</i>

7.3.1.3.4 Decreasing the Programmable Power Supply (PPS) Current

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when decreasing the current limit in the same *APDO*, not exceeding the minimum for that *APDO* and without changing the requested voltage is shown in [Figure 7.34, "Transition Diagram for decreasing the Current in PPS mode"](#). The sequence that **Shall** be followed is described in [Table 7.13, "Sequence Description for decreasing the Current in PPS mode"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.34, "Transition Diagram for decreasing the Current in PPS mode"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.34 Transition Diagram for decreasing the Current in PPS mode

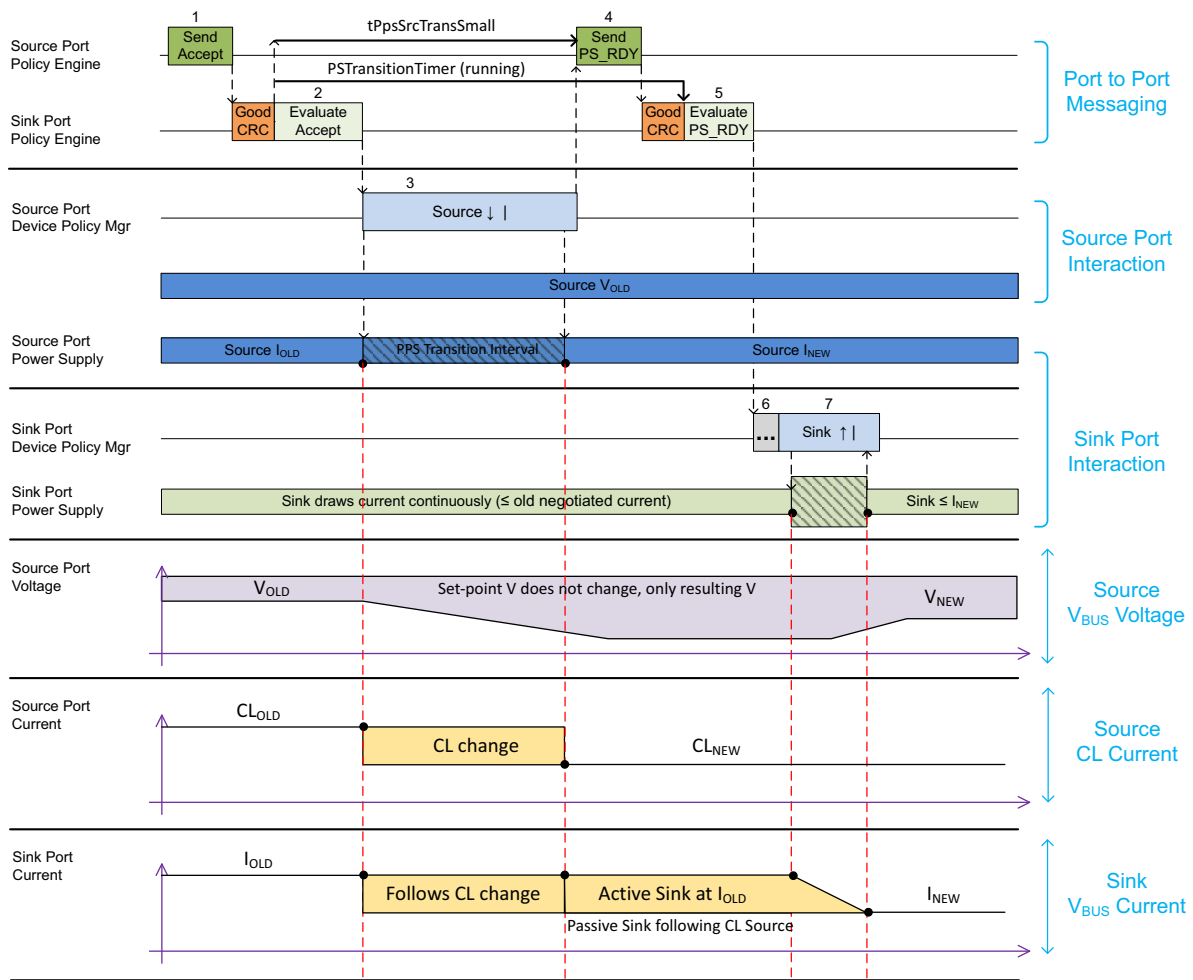


Table 7.13 Sequence Description for decreasing the Current in PPS mode

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to decrease its set-point for the current limit.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then evaluates the Accept Message and instructs the Sink to reduce its current to below the new <i>Negotiated</i> current level and starts the PSTransitionTimer.</i>
3	<i>The Power Supply decreases its <i>Current Limit</i> set-point to the new <i>Negotiated</i> value.</i>	<i>The Sink reduces its current to less than the new <i>Negotiated</i> current to prevent the Source from going into <i>Current Limit</i>.</i>
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tPpsSrcTransSmall of the end of the GoodCRC Message following the Accept Message.</i>	
5	<i>Policy Engine receives the GoodCRC Message from the Sink.</i>	<i>Policy Engine receives the PS_RDY Message.</i>
6		<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer and evaluates the PS_RDY Message.</i> <i>If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
7		<i>The Sink is allowed to draw I_{NEW} but must be aware the voltage on V_{BUS} can drop doing so.</i>

7.3.1.3.5 Same Request Programmable Power Supply (PPS)

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when the *Sink* requests the same voltage and current levels as the present *Negotiated* levels for voltage and current is shown in [Figure 7.35, "Transition Diagram for no change in Current or Voltage in PPS mode"](#). The sequence that **Shall** be followed is described in [Table 7.14, "Sequence Description for no change in Current or Voltage in PPS mode"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.35, "Transition Diagram for no change in Current or Voltage in PPS mode"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.35 Transition Diagram for no change in Current or Voltage in PPS mode

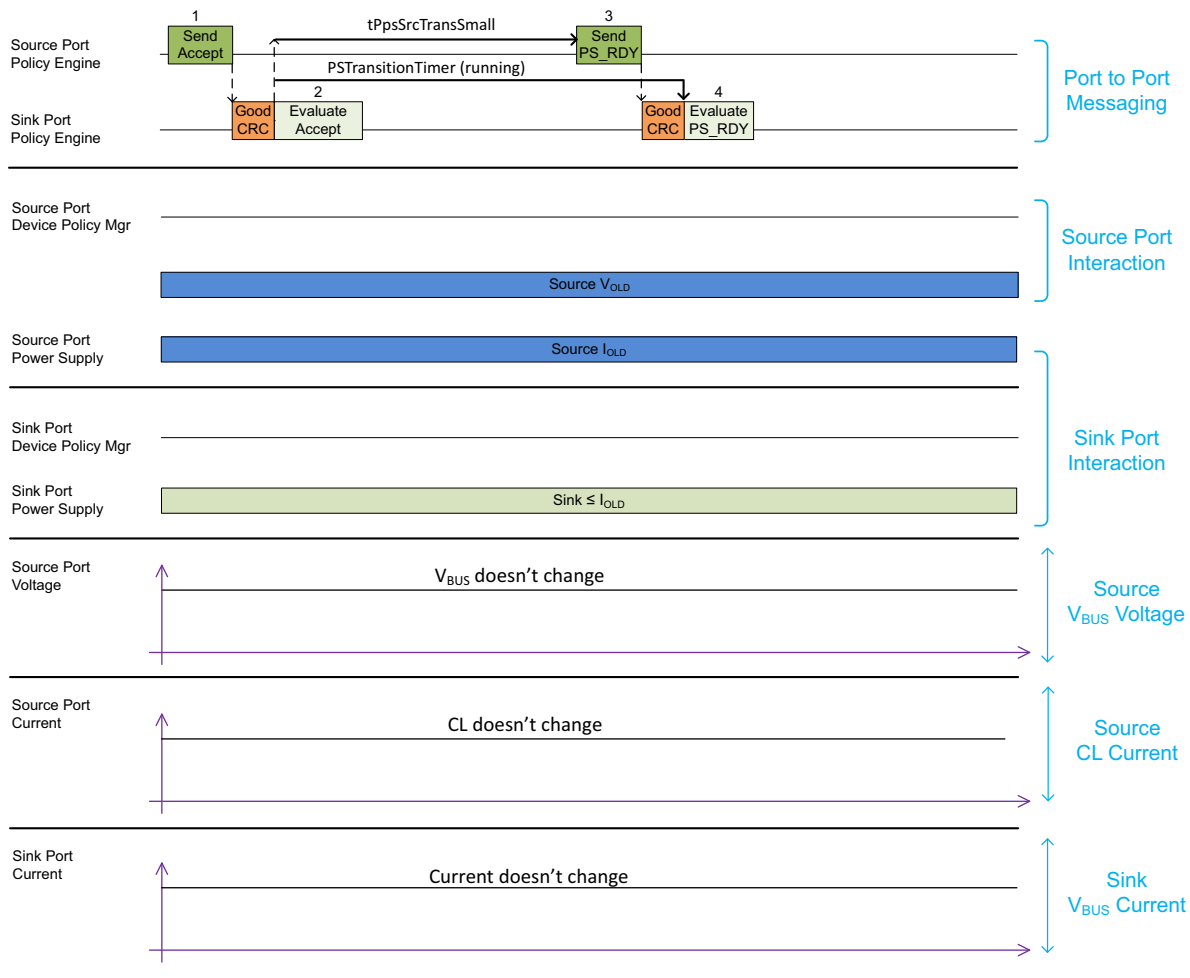


Table 7.14 Sequence Description for no change in Current or Voltage in PPS mode

Step	Source Port	Sink Port
1	Policy Engine sends the Accept Message to the Sink.	Policy Engine receives the Accept Message.
2	Protocol Layer receives the GoodCRC Message from the Sink.	Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then evaluates the Accept Message and starts the PSTransitionTimer .
3	The Policy Engine then sends the PS_RDY Message to the Sink starting within $t_{PpsSrcTransSmall}$ of the end of the GoodCRC Message following the Accept Message.	Policy Engine receives the PS_RDY Message.
4	Policy Engine receives the GoodCRC Message from the Sink. Note: The decision that no power transition is required could be made either by the Device Policy Manager or the power supply depending on implementation.	Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer and evaluates the PS_RDY Message from the Source. The Sink is already operating at the new power level, so no further action is required. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling .

7.3.1.4 Changing Voltage or Current within the same AVS APDO

7.3.1.4.1 Increasing the Adjustable Voltage Supply (AVS) Voltage

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed when increasing the voltage is shown in [Figure 7.36, "Transition Diagram for Increasing the Adjustable Power Supply Voltage"](#). The sequence that **shall** be followed is described in [Table 7.15, "Sequence Description for Increasing the Adjustable Voltage Supply Voltage"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.36, "Transition Diagram for Increasing the Adjustable Power Supply Voltage"](#), the Sink has previously sent a *Request Message* to the Source.

Figure 7.36 Transition Diagram for Increasing the Adjustable Power Supply Voltage

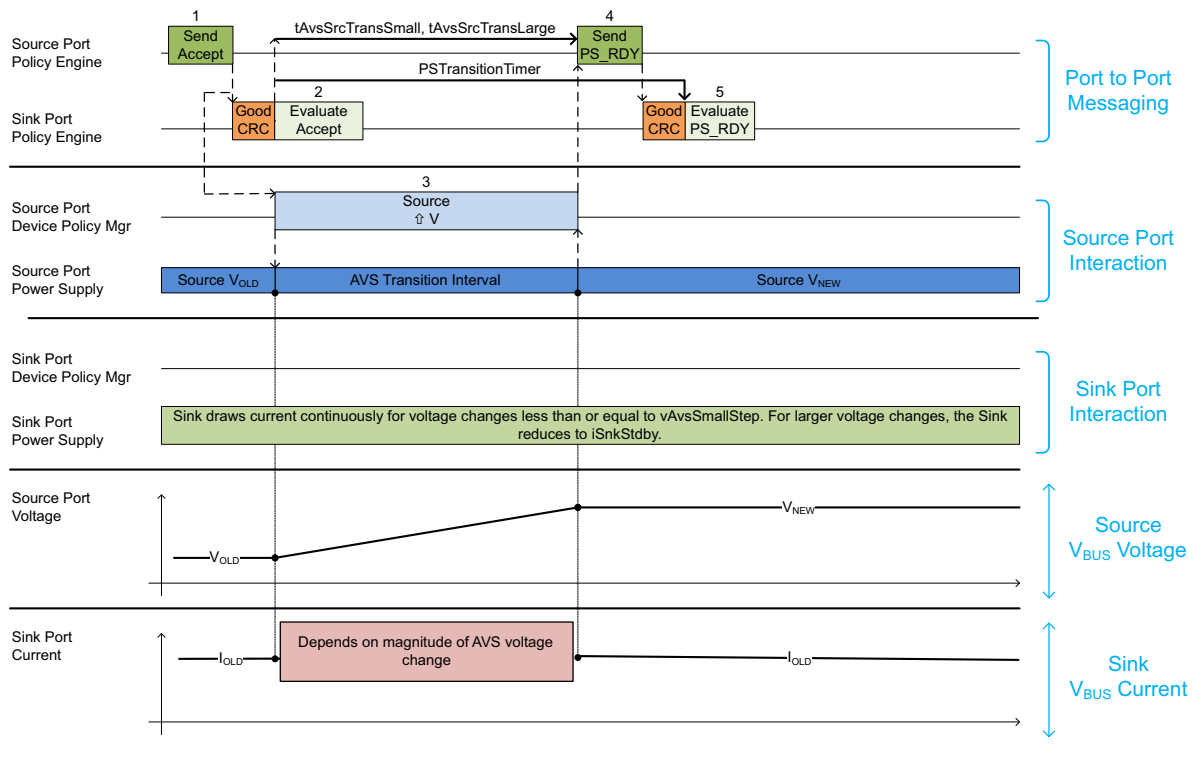


Table 7.15 Sequence Description for Increasing the Adjustable Voltage Supply Voltage

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to increase its output voltage.</i>	<p><i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i></p> <p><i>If the voltage increase is larger than vAvsSmallStep, the Sink Shall reduce its current draw to iSnkStdby within tSnkStdby. The reduction to iSnkStdby is not required if the voltage increase is less than or equal to vAvsSmallStep.</i></p>
3	<i>After sending the Accept Message, the AVS starts to increase its output voltage. The AVS new voltage set-point Shall be reached by tAvsSrcTransLarge for steps larger than vAvsSmallStep or else by tAvsSrcTransSmall. The power supply informs the Device Policy Manager that it has reached the new level. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tAvsSrcTransSmall or tAvsSrcTransLarge of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<p><i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new voltage set point. The Sink May begin operating at the new power level any time after evaluation of the PS_RDY Message.</i></p> <p><i>If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i></p>

7.3.1.4.2 Decreasing the Adjustable Voltage Supply (AVS) Voltage

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed when decreasing the voltage is shown in [Figure 7.37, "Transition Diagram for Decreasing the Adjustable Voltage Supply Voltage"](#). The sequence that **Shall** be followed is described in [Table 7.16, "Sequence Description for Decreasing the Adjustable Voltage Supply Voltage"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.37, "Transition Diagram for Decreasing the Adjustable Voltage Supply Voltage"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.37 Transition Diagram for Decreasing the Adjustable Voltage Supply Voltage

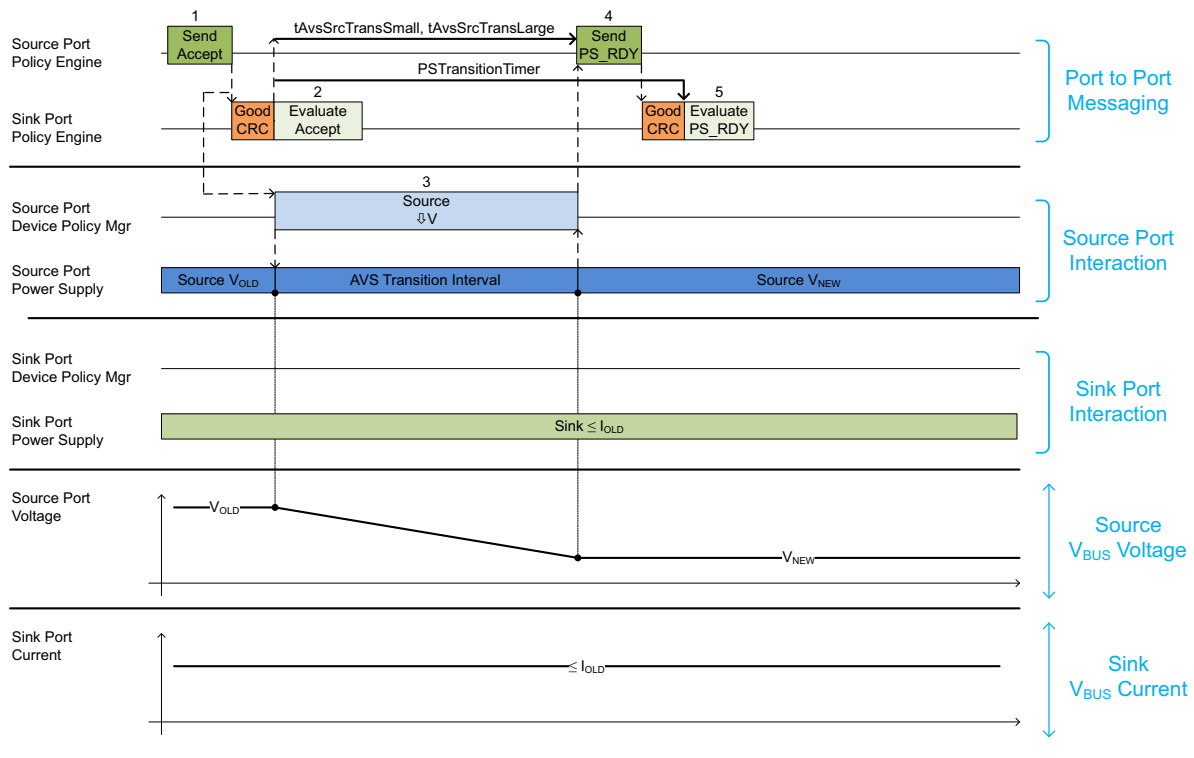


Table 7.16 Sequence Description for Decreasing the Adjustable Voltage Supply Voltage

Step	Source Port	Sink Port
1	<i>Policy Engine sends the Accept Message to the Sink.</i>	<i>Policy Engine receives the Accept Message.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to decrease its output voltage.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then starts the PSTransitionTimer and evaluates the Accept Message.</i>
3	<i>After sending the Accept Message, the AVS starts to decrease its output voltage. The AVS new voltage set-point Shall be reached by tAvsSrcTransLarge for steps larger than vAvsSmallStep or else by tAvsSrcTransSmall. The power supply informs the Device Policy Manager that it has reached the new level. The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the Sink starting within tAvsSrcTransSmall or tAvsSrcTransLarge of the end of the GoodCRC Message following the Accept Message.</i>	<i>The Policy Engine receives the PS_RDY Message from the Source.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PSTransitionTimer, evaluates the PS_RDY Message from the Source and tells the Device Policy Manager that the Source is operating at the new voltage set point (corresponding to vAvsNew). If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>

7.3.1.4.3 Same Request Adjustable Voltage Supply (AVS) Voltage

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed when the *Sink* requests the same voltage and current levels as the present *Negotiated* levels for voltage and current as shown in [Figure 7.38, "Transition Diagram for no change in Current or Voltage in AVS mode"](#). The sequence that **shall** be followed is described in [Table 7.17, "Sequence Description for no change in Current or Voltage in AVS mode"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.38, "Transition Diagram for no change in Current or Voltage in AVS mode"](#), the *Sink* has previously sent a *Request Message* to the *Source*.

Figure 7.38 Transition Diagram for no change in Current or Voltage in AVS mode

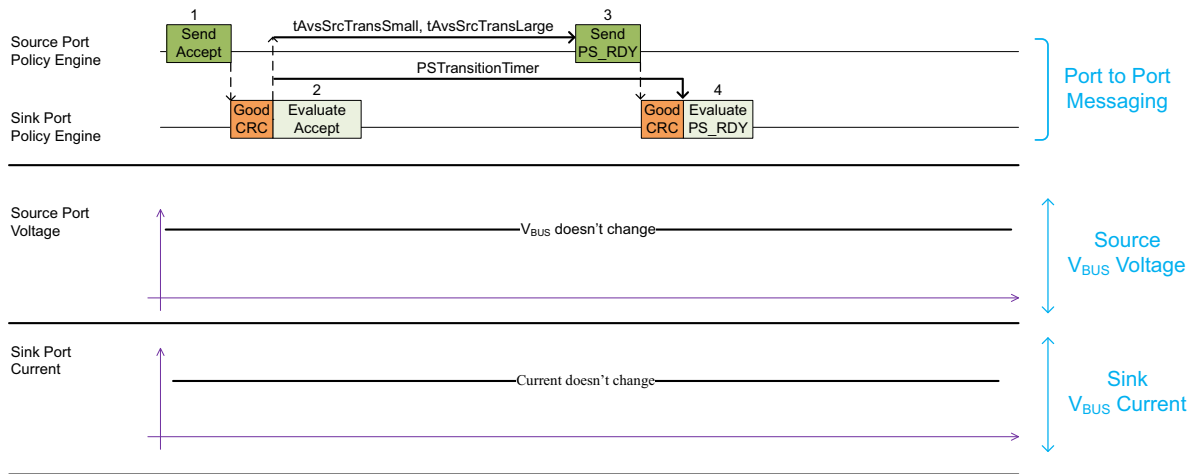


Table 7.17 Sequence Description for no change in Current or Voltage in AVS mode

Step	Source Port	Sink Port
1	Policy Engine sends the Accept Message to the Sink.	Policy Engine receives the Accept Message.
2	Protocol Layer receives the GoodCRC Message from the Sink.	Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then and starts the PStTransitionTimer and evaluates the Accept Message.
3	The Policy Engine sends the PS_RDY Message to the Sink starting within $t_{AvsSrcTransSmall}$ of the end of the GoodCRC Message following the Accept Message.	The Policy Engine receives the PS_RDY Message from the Source.
4	Protocol Layer receives the GoodCRC Message from the Sink. Note: The decision that no power transition is required could be made either by the <i>Device Policy Manager</i> or the power supply depending on implementation.	Protocol Layer sends the GoodCRC Message to the Source. Policy Engine then stops the PStTransitionTimer , evaluates the PS_RDY Message from the Source. The Sink is already operating at the new power level, so no further action is required. If the PS_RDY Message is not received before PStTransitionTimer times out the Sink sends Hard Reset Signaling.

7.3.2 Transitions Caused by Power Role Swap

7.3.2.1 Sink Requested Power Role Swap

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed during a *Sink* requested *Power Role Swap* is shown in [Figure 7.39, "Transition Diagram for a Sink Requested Power Role Swap"](#). The sequence that **shall** be followed is described in [Table 7.18, "Sequence Description for a Sink Requested Power Role Swap"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.39, "Transition Diagram for a Sink Requested Power Role Swap"](#), the *Sink* has previously sent a **PR_Swap** Message to the *Source*.

Figure 7.39 Transition Diagram for a Sink Requested Power Role Swap

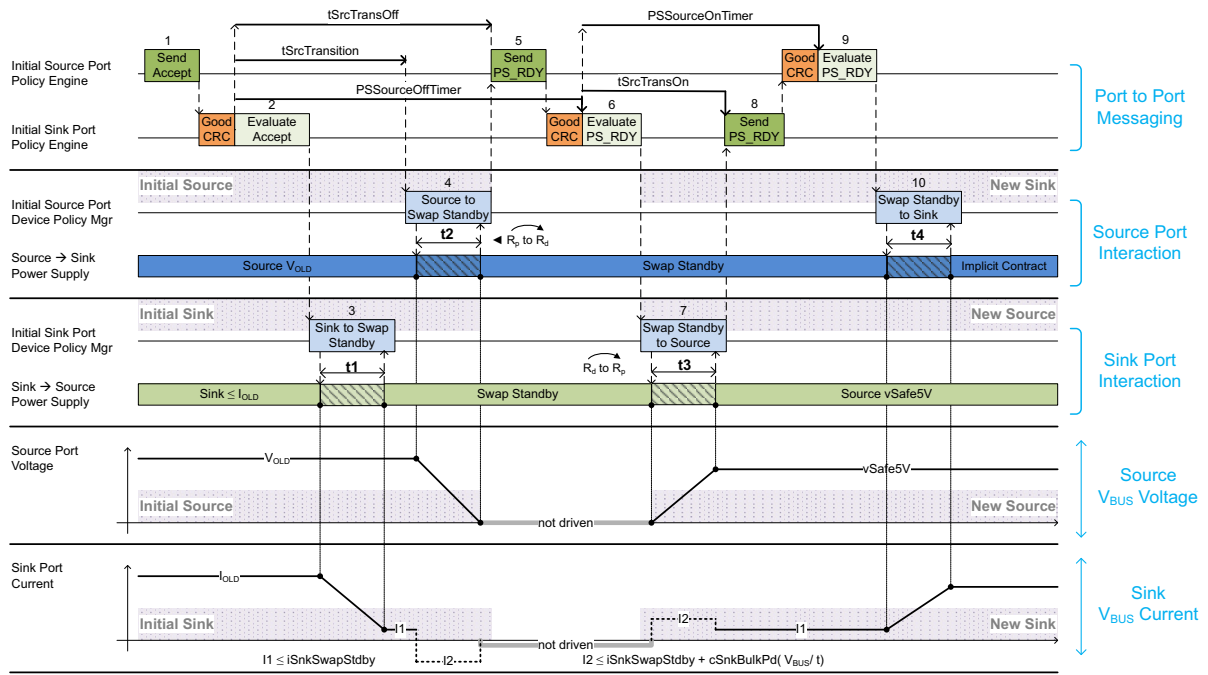


Table 7.18 Sequence Description for a Sink Requested Power Role Swap

Step	Initial Source Port → New Sink Port	Initial Sink Port → New Source Port
1	<i>Policy Engine sends the Accept Message to the Initial Sink.</i>	<i>Policy Engine receives the Accept.</i>
2	<i>Protocol Layer receives the GoodCRC Message from the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer sends the GoodCRC Message to the Initial Source. Policy Engine then starts the PSSourceOffTimer and evaluates the Accept Message.</i>
3		<i>Policy Engine tells the Device Policy Manager to instruct the power supply to transition to Swap Standby within tSnkStdby (t1); t1 shall complete before tSrcTransition min. When in Sink Standby the Initial Sink shall not draw more than iSnkSwapStdby (I1). The Sink shall not violate transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.</i>
4	<i>tSrcTransition after the GoodCRC Message was received, the power supply starts to change its output power capability to Swap Standby (see Section 7.1.10, "Swap Standby for Sources"). The power supply shall complete the transition to Swap Standby within tSrcSwapStdby (t2). The power supply informs the Device Policy Manager that it is ready to operate as the New Sink. The CC termination is changed from R_p to R_d (see [USB Type-C 2.4]). The power supply status is passed to the Policy Engine.</i>	
5	<i>The power supply is ready, and the Policy Engine sends the PS_RDY Message to the device that will become the New Source, starting within tSrcTransOff of the end of the GoodCRC Message following the Accept Message.</i>	
6	<i>Protocol Layer receives the GoodCRC Message from the device that will become the New Source. Policy Engine starts the PSSourceOnTimer. Upon sending the PS_RDY Message and receiving the GoodCRC Message the Initial Source is ready to be the New Sink.</i>	<i>The Protocol Layer sends the GoodCRC Message to the New Sink. Policy Engine stops the PSSourceOffTimer and tells the Device Policy Manager to instruct the power supply to operate as the New Source. If the PS_RDY Message is not received before PSTransitionTimer times out the Sink sends Hard Reset Signaling.</i>
7		<i>The CC termination is changed from R_d to R_p (see [USB Type-C 2.4]). The power supply as the New Source transitions from Swap Standby to sourcing default vSafe5V within tNewSrc (t3). The power supply informs the Device Policy Manager that it is operating as the New Source.</i>
8	<i>Policy Engine receives the PS_RDY Message from the Source.</i>	<i>Device Policy Manager informs the Policy Engine the power supply is ready, and the Policy Engine sends the PS_RDY Message to the New Sink, starting within tSrcTransOn of the end of the GoodCRC Message following the Accept Message.</i>

Table 7.18 Sequence Description for a Sink Requested Power Role Swap (Continued)

Step	Initial Source Port → New Sink Port	Initial Sink Port → New Source Port
9	<p><i>Protocol Layer</i> sends the GoodCRC Message to the <i>New Source</i> and then stops the PSSourceOnTimer.</p> <p><i>Policy Engine</i> evaluates the PS_RDY Message from the <i>New Source</i> and tells the <i>Device Policy Manager</i> to instruct the power supply to draw current as the <i>New Sink</i>.</p>	<p><i>Protocol Layer</i> receives the GoodCRC Message from the <i>New Sink</i>.</p>
10	<p>The power supply as the <i>New Sink</i> transitions from <i>Swap Standby</i> and begins to drawing the current allowed by the <i>Implicit Contract</i>. The power supply informs the <i>Device Policy Manager</i> that it is operating as the <i>New Sink</i>. At this point subsequent <i>Negotiations</i> between the <i>New Source</i> and the <i>New Sink</i> May proceed as normal. The <i>Sink</i> Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. The time duration (t4) depends on the magnitude of the load change (iLoadStepRate).</p>	

7.3.2.2 Source Requested Power Role Swap

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed during a *Source requested Power Role Swap* is shown in [Figure 7.40, "Transition Diagram for a Source Requested Power Role Swap"](#). The sequence that **shall** be followed is described in [Table 7.19, "Sequence Description for a Source Requested Power Role Swap"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Note: In [Figure 7.40, "Transition Diagram for a Source Requested Power Role Swap"](#), the *Source* has previously sent a **PR_Swap** Message to the *Sink*.

Figure 7.40 Transition Diagram for a Source Requested Power Role Swap

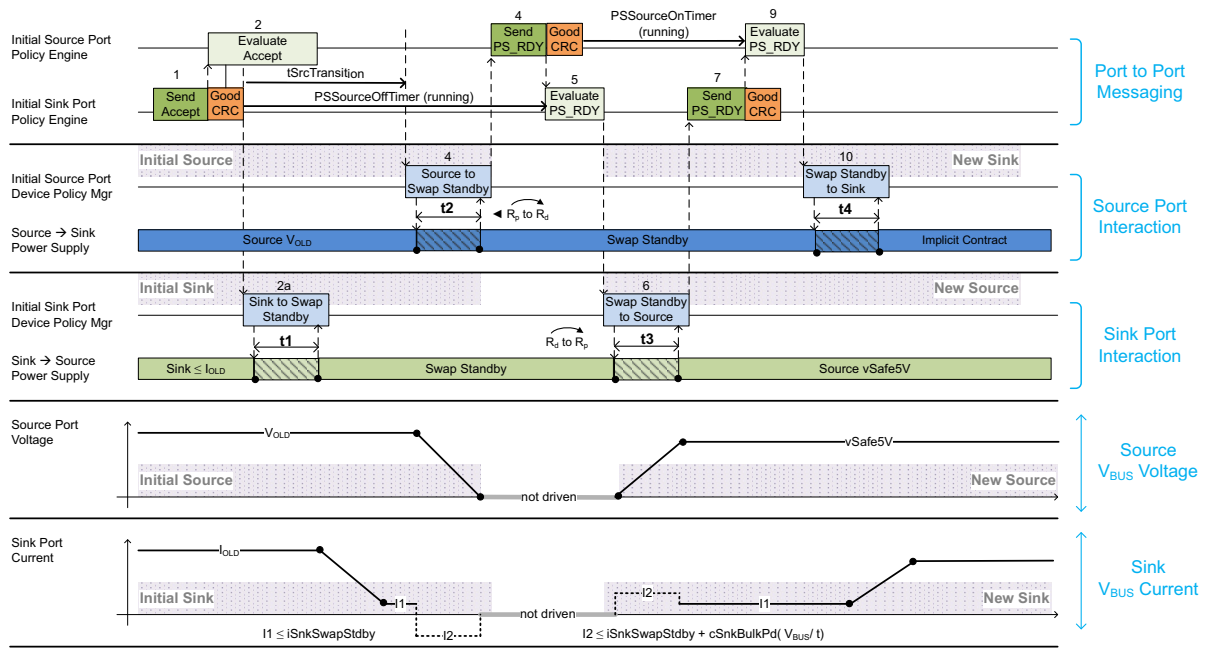


Table 7.19 Sequence Description for a Source Requested Power Role Swap

Step	Initial Source Port → New Sink Port	Initial Sink Port → New Source Port
1	<i>Policy Engine receives the Accept Message.</i>	<i>Policy Engine sends the Accept Message to the Initial Source.</i>
2	<i>Protocol Layer sends the GoodCRC Message to the Sink. The Policy Engine tells the Device Policy Manager to instruct the power supply to modify its output power.</i>	<i>Protocol Layer receives the GoodCRC Message from the Initial Source. Policy Engine starts the PSSourceOffTimer.</i>
2a		<i>The Policy Engine tells the Device Policy Manager to instruct the power supply to transition to Swap Standby. The power supply Shall complete the transition to Swap Standby within tSnkStdby (t1); t1 Shall complete before tSrcTransition. The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level. When in Sink Standby the Initial Sink Shall Not draw more than iSnkSwapStdby (I1).</i>
3	<i>tSrcTransition after the GoodCRC Message was sent the power supply starts to change its output power capability to Swap Standby (see Section 7.1.10, "Swap Standby for Sources"). The power supply Shall complete the transition to Swap Standby within tSrcSwapStdby (t2). The power supply informs the Device Policy Manager that it is ready to operate as the New Sink. The CC termination is changed from R_p to R_d (see [USB Type-C 2.4]). The power supply status is passed to the Policy Engine.</i>	
4	<i>The Policy Engine sends the PS_RDY Message to the device that will become the New Source, starting within tSrcTransOff of the end of the GoodCRC Message following the Accept Message.</i>	<i>Policy Engine receives the PS_RDY.</i>
5	<i>Protocol Layer receives the GoodCRC Message from the soon to be New Source. Policy Engine starts the PSSourceOnTimer. At this point the Initial Source is ready to be the New Sink.</i>	<i>Protocol Layer sends the GoodCRC Message to the New Sink. Policy Engine then stops the PSSourceOffTimer and tells the Device Policy Manager to instruct the power supply to operate as the New Source. If the PS_RDY Message is not received before the PSSourceOffTimer times out the Sink starts sending Hard Reset Signaling.</i>
6		<i>The CC termination is changed from R_d to R_p (see [USB Type-C 2.4]). The power supply as the New Source transitions from Swap Standby to sourcing default vSafe5V within tNewSrc (t3). The power supply informs the Device Policy Manager that it is operating as the New Source.</i>
7	<i>Policy Engine receives the PS_RDY Message.</i>	<i>Device Policy Manager informs the Policy Engine the power supply is ready, and the Policy Engine sends the PS_RDY Message to the New Sink, starting within tSrcTransOn of the end of the GoodCRC Message following the Accept Message.</i>

Table 7.19 Sequence Description for a Source Requested Power Role Swap (Continued)

Step	Initial Source Port → New Sink Port	Initial Sink Port → New Source Port
8	<p><i>Protocol Layer</i> sends the GoodCRC Message to the <i>New Source</i> and then stops the PSSourceOnTimer.</p> <p><i>Policy Engine</i> evaluates the PS_RDY Message from the <i>New Source</i> and tells the <i>Device Policy Manager</i> to instruct the power supply to draw current as the <i>New Sink</i>.</p>	<p><i>Protocol Layer</i> receives the GoodCRC Message from the <i>New Sink</i>.</p>
9	<p>The power supply as the <i>New Sink</i> transitions from <i>Swap Standby</i> to drawing the power allowed by the <i>Implicit Contract</i>. The power supply informs the <i>Device Policy Manager</i> that it is operating as the <i>New Sink</i>. At this point subsequent <i>Negotiations</i> between the <i>New Source</i> and the <i>New Sink</i> May proceed as normal. The <i>New Sink</i> Shall Not violate the transient load behavior defined in Section 7.2.6. "Transient Load Behavior" while transitioning to and operating at the new power level. The time duration (t4) depends on the magnitude of the load change (iLoadStepRate).</p>	

7.3.3 Transitions Caused by Hard Reset

7.3.3.1 Source Initiated Hard Reset

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed during a *Source Initiated Hard Reset* is shown in [Figure 7.41, "Transition Diagram for a Source Initiated Hard Reset"](#). The sequence that **Shall** be followed is described in [Table 7.20, "Sequence Description for a Source Initiated Hard Reset"](#). The timing parameters that **Shall** be applied are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Figure 7.41 Transition Diagram for a Source Initiated Hard Reset

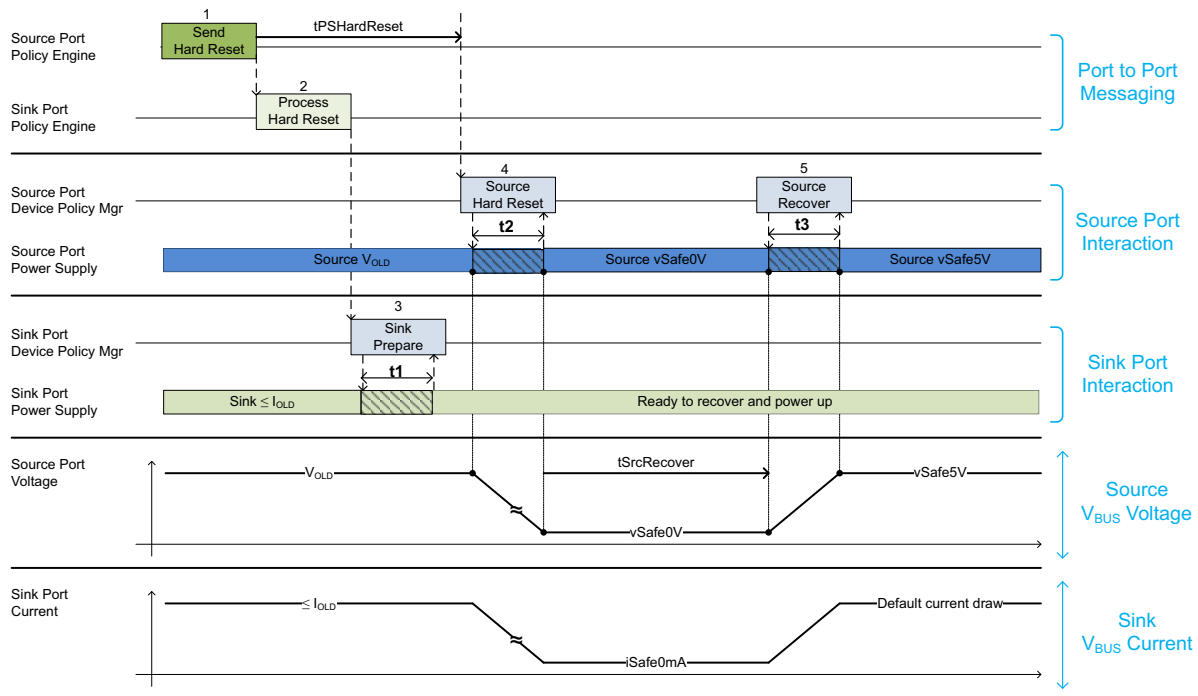


Table 7.20 Sequence Description for a Source Initiated Hard Reset

Step	Source Port	Sink Port
1	Policy Engine sends Hard Reset Signaling to the Sink.	Sink receives Hard Reset Signaling.
2		Policy Engine is informed of the Hard Reset. Policy Engine tells the Device Policy Manager to instruct the power supply to prepare for a Hard Reset.
3		The Sink prepares for the Hard Reset within tSinkHardResetPrepare (t1) and passes an indication to the Device Policy Manager. The Sink Shall Not draw more than iSafe0mA when V_{BUS} is driven to vSafe0V .
4	Policy Engine waits tPSHardReset after sending Hard Reset Signaling and then tells the Device Policy Manager to instruct the power supply to perform a Hard Reset. The transition to vSafe0V Shall occur within tSafe0V (t2).	
5	After tSrcRecover the Source applies power to V_{BUS} in an attempt to re-establish communication with the Sink and resume USB Default Operation. The transition to vSafe5V Shall occur within tSrcTurnOn (t3).	The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.

7.3.3.2 Sink Initiated Hard Reset

The interaction of the *System Policy*, *Device Policy*, and power supply that **Shall** be followed during a *Sink Initiated Hard Reset* is shown in [Figure 7.42, "Transition Diagram for a Sink Initiated Hard Reset"](#). The sequence that **Shall** be followed is described in [Table 7.21, "Sequence Description for a Sink Initiated Hard Reset"](#). The timing parameters that **Shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Figure 7.42 Transition Diagram for a Sink Initiated Hard Reset

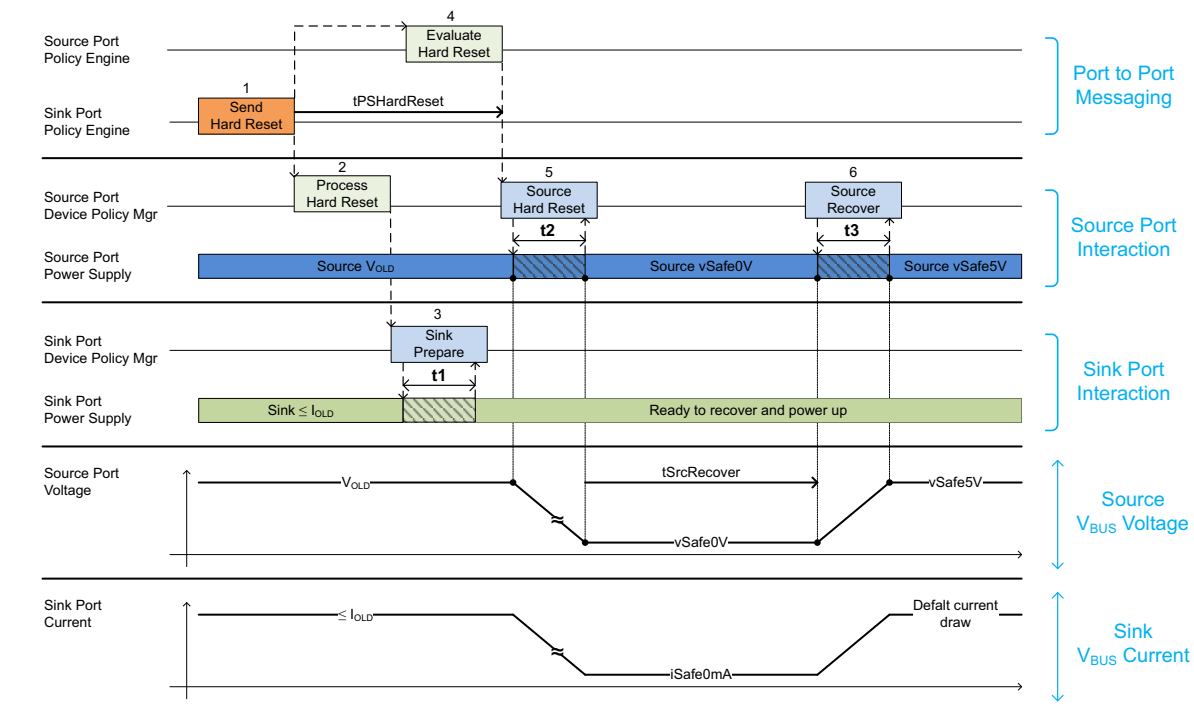


Table 7.21 Sequence Description for a Sink Initiated Hard Reset

Step	Source Port	Sink Port
1		Policy Engine sends Hard Reset Signaling to the Source.
2		Policy Engine tells the Device Policy Manager to instruct the power supply to prepare for a Hard Reset.
3		The Sink prepares for the Hard Reset within tSinkHardResetPrepare (t1) and passes an indication to the Device Policy Manager. The Sink Shall Not draw more than iSafe0mA when V_{BUS} is driven to vSafe0V .
4	Policy Engine is informed of the Hard Reset.	
5	Policy Engine waits tPSHardReset after receiving Hard Reset Signaling and then tells the Device Policy Manager to instruct the power supply to perform a Hard Reset. The transition to vSafe0V Shall occur within tSafe0V (t2).	
6	After tSrcRecover the Source applies power to V_{BUS} in an attempt to re-establish communication with the Sink and resume USB Default Operation. The transition to vSafe5V Shall occur within tSrcTurnOn (t3).	The Sink Shall Not violate the transient load behavior defined in Section 7.2.6, "Transient Load Behavior" while transitioning to and operating at the new power level.

7.3.4 Transitions Caused by Fast Role Swap

7.3.4.1 Fast Role Swap

The interaction of the *System Policy*, *Device Policy*, and power supply that **shall** be followed during a *Fast Role Swap* is shown in [Figure 7.43, "Transition Diagram for Fast Role Swap"](#). The parallel sequences that **shall** be followed are described in [Table 7.22, "Sequence Description for Fast Role Swap"](#). The timing parameters that **shall** be followed are listed in [Table 7.23, "Source Electrical Parameters"](#), [Table 7.24, "Sink Electrical Parameters"](#), and [Table 7.25, "Common Source/Sink Electrical Parameters"](#). Negotiations between the *New Source* and the *New Sink* **may** occur after the *New Source* sends the final **PS_RDY** Message.

Note: In [Figure 7.43, "Transition Diagram for Fast Role Swap"](#) and [Table 7.22, "Sequence Description for Fast Role Swap"](#) numbers are used to indicate *Message* related steps and letters are used to indicate other events.

Figure 7.43 Transition Diagram for Fast Role Swap

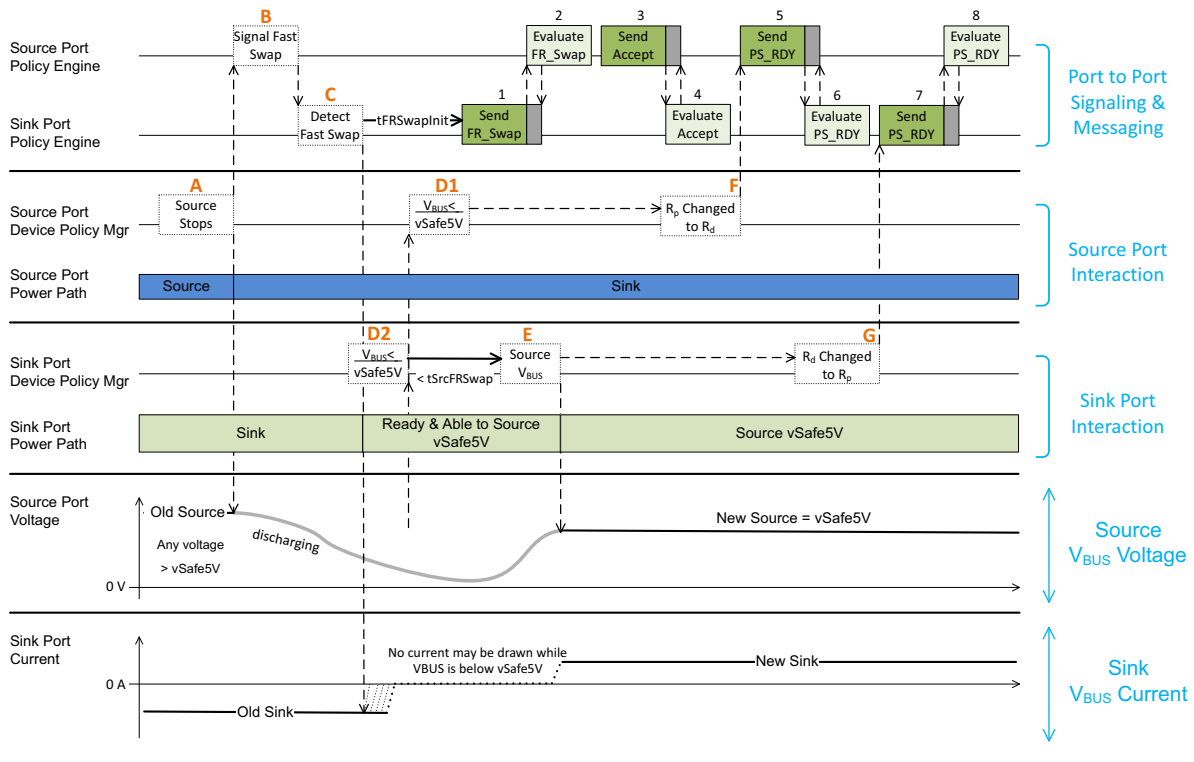


Table 7.22 Sequence Description for Fast Role Swap

Step	Initial Source Port → New Sink Port	Initial Sink Port → New Source Port
Fast Role Swap Request and Power Transition		
A	The <i>Source</i> connected to the <i>Hub UFP</i> (see Figure 7.16, "VBUS Power during Fast Role Swap") stops sourcing <i>VBUS</i> .	
B	<i>Policy Engine</i> sends the <i>Fast Role Swap Request</i> to the <i>Initial Sink</i> on the <i>CC</i> wire. When $VBUS < vSafe5V$ (min), it tells the <i>Device Policy Manager</i> not to draw more than <i>iSnkStdby</i> until the <i>tSnkFRSwap</i> timer has elapsed.	
C		<i>Policy Engine</i> detects the <i>Fast Role Swap Request</i> on the <i>CC</i> wire from the <i>Initial Source</i> and shall send the <i>FR_Swap</i> Message back to the <i>Initial Source</i> (that is no longer powering <i>VBUS</i>) within time <i>tFRSwapInit</i> .
D1	The <i>Policy Engine</i> monitors for $VBUS \leq vSafe5V$ so that a <i>PS_RDY</i> Message can be sent to the <i>New Source</i> at Step 5 of the messaging sequence.	
D2		The <i>Policy Engine</i> monitors for $VBUS \leq vSafe5V$ so the <i>Initial Sink</i> can assume the <i>Power Role</i> of <i>New Source</i> and begin to source <i>VBUS</i> .
E		When $VBUS = vSafe5V$ the <i>New Source</i> may provide power to <i>VBUS</i> . When $VBUS < vSafe5V$ the <i>New Source</i> shall provide power to <i>VBUS</i> within <i>tSrcFRSwap</i> . Once the <i>New Source</i> is providing power, the <i>PS_RDY</i> Message can be sent to the <i>New Sink</i> at Step 7 of the messaging sequence.
F	The <i>CC</i> termination is changed from R_p to R_d (see [USB Type-C 2.4]) before the <i>New Sink</i> sends the <i>PS_RDY</i> Message at Step 5 to the <i>New Source</i> .	
G		The <i>CC</i> termination is changed from R_d to R_p (see [USB Type-C 2.4]) before the <i>New Source</i> sends the <i>PS_RDY</i> Message at Step 7 to the <i>New Sink</i> .
Fast Role Swap Message Sequence		
1	<i>Policy Engine</i> receives the <i>FR_Swap</i> Message from the <i>Initial Sink</i> that is transitioning to be the <i>New Source</i> .	<i>Policy Engine</i> sends the <i>FR_Swap</i> Message to the <i>Initial Source</i> (that is no longer powering <i>VBUS</i>) after detecting the <i>Fast Role Swap Request</i> at Step C.
2	<i>Protocol Layer</i> sends the <i>GoodCRC</i> Message to the <i>Initial Sink</i> . <i>Policy Engine</i> then evaluates the <i>FR_Swap</i> Message.	<i>Protocol Layer</i> receives the <i>GoodCRC</i> Message from the <i>Initial Source</i> .
3	<i>Policy Engine</i> sends an <i>Accept</i> Message to the <i>Initial Sink</i> that is transitioning to be the <i>New Source</i> .	<i>Policy Engine</i> receives the <i>Accept</i> Message from the <i>Initial Source</i> that is transitioning to be the <i>New Sink</i> .
4	<i>Protocol Layer</i> receives the <i>GoodCRC</i> Message from the <i>Initial Sink</i> that is transitioning to be the <i>New Source</i> .	<i>Protocol Layer</i> sends the <i>GoodCRC</i> Message to the <i>Initial Source</i> that is transitioning to be the <i>New Sink</i> .

Table 7.22 Sequence Description for Fast Role Swap (Continued)

Step	Initial Source Port → New Sink Port	Initial Sink Port → New Source Port
5	<i>Policy Engine sends a PS_RDY Message to the Initial Sink that is transitioning to be the New Source. The Policy Engine Shall start the PS_RDY Message at least tFRSwap5V after it has sent the Accept Message, and when Step D1 has also been completed.</i>	<i>Policy Engine receives the PS_RDY Message from the New Sink.</i>
6	<i>Protocol Layer receives the GoodCRC Message from the New Source.</i>	<i>Protocol Layer sends the GoodCRC Message from the Initial Sink that has completed the transition to New Source. Policy Engine then evaluates the PS_RDY Message.</i>
7	<i>Policy Engine receives the PS_RDY Message from the New Source.</i>	<i>Policy Engine sends a PS_RDY Message to the New Sink. The Policy Engine Shall wait for Step E before sending the PS_RDY Message, and Shall send the PS_RDY Message within tFRSwapComplete of receiving the PS_RDY Message from the Initial Source Port.</i>

7.4 Electrical Parameters

7.4.1 Source Electrical Parameters

The *Source* Electrical Parameters that **shall** be followed are specified in [Table 7.23, "Source Electrical Parameters"](#).

Table 7.23 Source Electrical Parameters

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>cSrcBulk</i>	Source bulk capacitance when a <i>Port</i> is powered from a dedicated supply. ¹	10			μF	Section 7.1.2
<i>cSrcBulkShared</i>	Source bulk capacitance when a <i>Port</i> is powered from a shared supply. ¹	120			μF	Section 7.1.2
<i>DNL</i> (Differential Non-Linearity)	Deviation between ideal analog values corresponding to adjacent input digital values	-1	0	+1	LSB	Section 7.1.4.2.1
<i>iPpsCLMin</i>	SPR PPS Minimum Current Limit setting.	1			A	Section 7.1.4.2.2
<i>iPpsCLNew</i>	Current Limit accuracy					Section 7.1.4.2.2
	1A ≤ Operating Current ≤ 3A	-150		150	mA	
	Operating current > 3A	-5		5	%	
<i>iPpsCLStep</i>	SPR PPS Current Limit programming step size (1 LSB).		50		mA	Section 7.1.4.2.2
<i>iPpsCLLoadReleaseRate</i>	Maximum load decrease slew rate during Current Limit set-point changes.	-150			mA/μs	Section 7.1.4.2.2
<i>iPpsCLLoadStepRate</i>	Maximum load increase slew rate during Current Limit set-point changes.			150	mA/μs	Section 7.1.4.2.2
<i>iPpsCLTransient</i>	Allowed output current overshoot when a load increase occurs while in CL mode.			New load + 100	mA	Section 7.1.4.2.2
	Allowed output current undershoot when a load decrease occurs while in CL mode.	New load – 100			mA	

1) The *Source* **shall** charge and discharge the total bulk capacitance to meet the transition time requirements.

Table 7.23 Source Electrical Parameters (Continued)

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>iPpsCVCLTransient</i>	CV to CL transient current bounds assuming the Operating Voltage reduction of Section 7.2.3.1, "Programmable Power Supply Sink Standby" .	<i>iPpsCLNew</i> - 100		New load + 500	mA	Section 7.1.4.2.2
<i>tAvsTransient</i>	The maximum time for the AVS to be between <i>vAvsNew</i> and <i>vAvsValid</i> in response to a load transient.			5	ms	Section 7.1.8.4
<i>tAvsSrcTransLarge</i>	The time the AVS set-point shall transition between requested voltages for steps larger than <i>vAvsSmallStep</i> .	0		700	ms	Section 7.1.4.3.1
<i>tAvsSrcTransSmall</i>	The time the AVS set-point shall transition between requested voltages for steps smaller than <i>vAvsSmallStep</i> .	0		50	ms	Section 7.1.4.3.1
<i>tNewSnk</i>	Time allowed for an <i>Initial Source</i> in <i>Swap Standby</i> to transition <i>New Sink</i> operation.			15	ms	Section 7.1.10 Figure 7.39 Figure 7.40
<i>tPpsCLCVTransient</i>	CL to CV transient voltage settling time.			275	ms	Section 7.1.4.2.2
<i>tPpsCLProgramSettle</i>	SPR PPS Current Limit programming settling time.			250	ms	Section 7.1.4.2.2
<i>tPpsCLSettle</i>	CL load transient current settling time.			250	ms	Section 7.1.4.2.2
<i>tPpsCVCLTransient</i>	CV to CL transient settling time.			250	ms	Section 7.1.8.3
<i>tPpsSrcTransLarge</i>	The time the Programmable Power Supply's set-point shall transition between requested voltages for steps larger than <i>vPpsSmallStep</i> .	0		275	ms	Section 7.3.1.3
1) The Source shall charge and discharge the total bulk capacitance to meet the transition time requirements.						

Table 7.23 Source Electrical Parameters (Continued)

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>tPpsSrcTransSmall</i>	The time the Programmable Power Supply's set-point shall transition between requested voltages for steps less than or equal to <i>vPpsSmallStep</i> .	0		25	ms	Section 7.3.1.3
<i>tPpsTransient</i>	The maximum time for the Programmable Power Supply to be between <i>vPpsNew</i> and <i>vPpsValid</i> in response to a load transient when target load is greater than or equal to 60mA.			5	ms	Section 7.1.8.3
	The maximum time for the Programmable Power Supply to be between <i>vPpsNew</i> and <i>vPpsValid</i> in response to a load transient when target load is less than 60mA.			150	ms	Section 7.1.8.3
<i>tSrcFRSwap</i>	Time from the <i>Initial Sink</i> detecting that <i>VBUS</i> has dropped below <i>vSafe5V</i> until the <i>Initial Sink</i> /new <i>Source</i> is able to supply <i>USB Type-C</i> Current (see [USB Type-C 2.4])			150	μs	Section 7.1.13
1) The Source shall charge and discharge the total bulk capacitance to meet the transition time requirements.						

Table 7.23 Source Electrical Parameters (Continued)

Parameter		Description	MIN	TYP	MAX	UNITS	Reference
<i>tSrcReady</i>	<i>SPR Mode</i>	Time from positive/negative transition start (t0) to when the <i>Source</i> is ready to provide the newly <i>Negotiated</i> power level. Applies only to <i>SPR Mode</i> voltage transitions.			285	ms	Figure 7.2 Figure 7.3
	<i>EPR Mode</i>	Time from positive/negative transition start (t0) to when the <i>Source</i> is ready to provide the newly <i>Negotiated</i> power level. Applies to <i>EPR Mode</i> voltage transitions and any voltage transition that either begins or ends in <i>EPR Mode</i> .			720		
<i>tSrcRecover</i>	<i>SPR Mode</i>	Time allotted for the <i>Source</i> to recover.	0.66		1.0	s	Section 7.1.5
	<i>EPR Mode</i>		1.085		1.425		
<i>tSrcSettle</i>	<i>SPR Mode</i>	Time from positive/negative transition start (t0) to when the transitioning voltage is within the range <i>vSrcNew</i> . Applies only to <i>SPR Mode</i> voltage transitions.			275	ms	Figure 7.2
	<i>EPR Mode</i>	Time from positive/negative transition start (t0) to when the transitioning voltage is within the range <i>vAvsNew</i> . Applies to <i>EPR Mode</i> voltage transitions and any voltage transition that either begins or ends in <i>EPR Mode</i> .			700		
<i>tSrcSwapStdby</i>		The maximum time for the <i>Source</i> to transition to <i>Swap Standby</i> .			650	ms	Section 7.1.10 Figure 7.17 Figure 7.18
1) The <i>Source</i> Shall charge and discharge the total bulk capacitance to meet the transition time requirements.							

Table 7.23 Source Electrical Parameters (Continued)

Parameter		Description	MIN	TYP	MAX	UNITS	Reference
<i>tSrcTransient</i>		The maximum time for the <i>Source</i> output voltage to be between <i>vSrcNew</i> and <i>vSrcValid</i> in response to a load transient when target load is greater or equal to than 60mA.			5	ms	Section 7.1.8
		The maximum time for the <i>Source</i> output voltage to be between <i>vSrcNew</i> and <i>vSrcValid</i> in response to a load transient when target load is less than 60mA.			150	ms	Section 7.1.8
<i>tSrcTransition</i>		The time the <i>Source</i> Shall wait before transitioning the power supply to ensure that the <i>Sink</i> has sufficient time to prepare (does not apply to transitions within the same PPS or AVS APDO).	25		35	ms	Section 7.3
<i>tSrcTransOff</i>	<i>SPR Mode</i>	Time from the last bit of the <i>GoodCRC Message</i> acknowledging the <i>Accept Message</i> in response to the <i>PR_Swap Message</i> until the <i>PS_RDY Message</i> must be started. Applies only to <i>SPR Mode</i> voltage transitions.			690	ms	Section 7.3.2
<i>tSrcTransOn</i>		Time from the last bit of the <i>GoodCRC Message</i> acknowledging the <i>PS_RDY Message</i> sent by the new <i>Source</i> , in response to the <i>PR_Swap Message</i> until the <i>PS_RDY Message</i> must be started.			280	ms	Section 7.3.2
1) The <i>Source</i> Shall charge and discharge the total bulk capacitance to meet the transition time requirements.							

Table 7.23 Source Electrical Parameters (Continued)

Parameter		Description	MIN	TYP	MAX	UNITS	Reference
<i>tSrcTransReq</i>	<i>SPR Mode</i>	Time from the last bit of the GoodCRC Message acknowledging the Accept Message in response to the Request Message until the PS_RDY Message must be started. Applies only to <i>SPR Mode</i> voltage transitions.			325	ms	Section 7.3
	<i>EPR Mode</i>	Time from the last bit of the GoodCRC Message acknowledging the Accept Message in response to the Request Message until the PS_RDY Message must be started. Applies to <i>EPR Mode</i> voltage transitions and any voltage transition that either begins or ends in <i>EPR Mode</i> .			760	ms	Section 7.3
<i>tSrcTurnOn</i>		Transition time from vSafe0V to vSafe5V .			275	ms	Figure 7.10 Table 7.20 Table 7.21
<i>vAvsMaxVoltage</i>		Maximum Voltage Field in the <i>AVS APDO</i> .	<i>APDO</i> Max Voltage *0.95		<i>APDO</i> Max Voltage * 1.05	V	Section 7.1.4.3.1
<i>vAvsMinVoltage</i>		Minimum Voltage Field in the <i>AVS APDO</i> .	<i>APDO</i> Min Voltage *0.95		<i>APDO</i> Min Voltage * 1.05	V	Section 7.1.4.3.1
<i>vAvsNew</i>		Adjustable <i>RDO</i> Output Voltage measured at the <i>Source</i> receptacle.	<i>RDO</i> Output Voltage *0.95	<i>RDO</i> Output Voltage	<i>RDO</i> Output Voltage *1.05	V	Section 7.1.8.4
<i>vAvsSlewNeg</i>		<i>AVS</i> maximum slew rate for negative voltage changes.			-30	mV/μs	Section 7.1.8.4

1) The *Source* **Shall** charge and discharge the total bulk capacitance to meet the transition time requirements.

Table 7.23 Source Electrical Parameters (Continued)

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>vAvsSlewPos</i>	AVS maximum slew rate for positive voltage changes.			30	mV/μs	Section 7.1.8.4
<i>vAvsSmallStep</i>	AVS step size defined as a small step relative to the previous <i>vAvsNew</i> .	-1.0		1.0	V	Section 7.1.4.3.1
<i>vAvsStep</i>	AVS voltage programming step size.		100		mV	Section 7.1.8.4
<i>vAvsValid</i>	The range in addition to <i>vAvsNew</i> which the AVS output is considered Valid during and after a transition as well as in response to a transient load condition.	-0.5		0.5	V	Section 7.1.8.4
<i>vPpsCLCVTransient</i>	CL to CV load transient voltage bounds.	Operating Voltage * 0.95 – 0.1V		Operating Voltage * 1.05 + 0.1V	V	Section 7.1.4.2.2
<i>vPpsMaxVoltage</i>	Maximum Voltage Field in the Programmable Power Supply <i>APDO</i> .	<i>APDO</i> Max Voltage *0.95		<i>APDO</i> Max Voltage * 1.05	V	Section 7.1.4.2.1
<i>vPpsMinVoltage</i>	Minimum Voltage Field in the Programmable Power Supply <i>APDO</i> .	<i>APDO</i> Min Voltage *0.95		<i>APDO</i> Min Voltage * 1.05	V	Section 7.1.4.2.1
<i>vPpsNew</i>	Programmable <i>RDO</i> Output Voltage measured at the <i>Source</i> receptacle.	<i>RDO</i> Output Voltage *0.95	<i>RDO</i> Output Voltage	<i>RDO</i> Output Voltage *1.05	V	Section 7.1.8.3
<i>vPpsShutdown</i>	The voltage at which the <i>SPR PPS</i> shuts down when operating in <i>CL</i> .	<i>APDO</i> Minimum Voltage * 0.85		<i>APDO</i> Minimum Voltage * 0.95	V	Section 7.1.4.2.2
<i>vPpsSlewNeg</i>	Programmable Power Supply maximum slew rate for negative voltage changes			-30	mV/μs	Section 7.1.8.3
<i>vPpsSlewPos</i>	Programmable Power Supply maximum slew rate for positive voltage changes			30	mV/μs	Section 7.1.8.3

1) The *Source* **Shall** charge and discharge the total bulk capacitance to meet the transition time requirements.

Table 7.23 Source Electrical Parameters (Continued)

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>vPpsSmallStep</i>	PPS Step size defined as a small step relative to the previous <i>vPpsNew</i> .	-500		500	mV	Section 7.1.4.2.2
<i>vPpsStep</i>	PPS voltage programming step size (1 <i>LSB</i>).		20		mV	Section 7.1.8.3
<i>vPpsValid</i>	The range in addition to <i>vPpsNew</i> which the Programmable Power Supply output is considered Valid in response to a load step.	-0.1		0.1	V	Section 7.1.8.3
<i>vSmallStep</i>	<i>V_{BUS}</i> step size increase defined as a small step relative to the previous <i>V_{BUS}</i> when Requesting a different (<i>A</i>) <i>PDO</i> .			500	mV	Section 7.1.4.3.1
<i>vSrcNeg</i>	Most negative voltage allowed during transition.			-0.3	V	Figure 7.10
<i>vSrcNew</i>	<i>Fixed Supply</i> output measured at the <i>Source</i> receptacle.	PDO Voltage *0.95	PDO Voltage	PDO Voltage *1.05	V	Table 7.2
	<i>Variable Supply</i> output measured at the <i>Source</i> receptacle.	PDO Minimum Voltage		PDO Maximum Voltage	V	
	<i>Battery Supply</i> output measured at the <i>Source</i> receptacle.	PDO Minimum Voltage		PDO Maximum Voltage	V	
<i>vSrcPeak</i>	The range that a <i>Fixed Supply</i> or <i>EPR AVS</i> in Peak Current operation is allowed when overload conditions occur.	PDO Voltage *0.90		PDO Voltage *1.05	V	Table 6.10 Table 6.16 Figure 7.14
<i>vSrcSlewNeg</i>	Maximum slew rate allowed for negative voltage transitions. Limits current based on a 3 A connector rating and maximum <i>Sink</i> bulk capacitance of 100 μ F.			-30	mV/ μ s	Section 7.1.4.2 Table 7.2

1) The *Source* **Shall** charge and discharge the total bulk capacitance to meet the transition time requirements.

Table 7.23 Source Electrical Parameters (Continued)

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>vSrcSlewPos</i>	Maximum slew rate allowed for positive voltage transitions. Limits current based on a 3 A connector rating and maximum <i>Sink</i> bulk capacitance of 100 μ F.			30	mV/ μ s	Section 7.1.4 Figure 7.2
<i>vSrcValid</i>	The range in addition to <i>vSrcNew</i> which a newly <i>Negotiated</i> voltage is considered Valid during and after a transition as well as in response to a transient load condition. This range also applies to <i>vSafe5V</i> .	-0.5		0.5	V	Figure 7.2 Figure 7.3 Section 7.1.8
1) The Source Shall charge and discharge the total bulk capacitance to meet the transition time requirements.						

7.4.2 Sink Electrical Parameters

The *Sink* Electrical Parameters that **shall** be followed are specified in [Table 7.24, "Sink Electrical Parameters"](#).

Table 7.24 Sink Electrical Parameters

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>cSnkBulk</i>	<i>Sink</i> bulk capacitance on <i>V_{BUS}</i> at <i>Attach</i> and during <i>FRS</i> after the <i>Initial Source</i> stops sourcing and prior to establishing the <i>First Explicit Contract</i> (see Appendix E, "FRS System Level Example" for an example). ¹			See [USB 3.2]		Section 7.2.2 [USB 3.2]
<i>cSnkBulkPd</i>	Bulk capacitance on <i>V_{BUS}</i> a <i>Sink</i> is allowed after a successful <i>Negotiation</i> . ¹			100	μF	Section 7.2.2
<i>iLoadReleaseRate</i>	Load release di/dt.	-150			mA/ μs	Section 7.2.6
<i>iLoadStepRate</i>	Load step di/dt.			150	mA/ μs	Section 7.2.6
<i>iNewFrsSink</i>	Maximum current the <i>New Sink</i> can draw during a <i>Fast Role Swap</i> until the <i>New Source</i> applies <i>R_p</i> . Matches the required <i>Fast Role Swap required USB Type-C Current</i> Current field of the <i>Fixed Supply</i> PDO of the <i>Initial Source's Sink Capabilities</i> Message.			Default USB current or 1.5 or 3.0	A	Section 7.1.13
<i>iOvershoot</i>	Positive or negative overshoot when a load change occurs less than or equal to <i>iLoadStepRate</i> ; relative to the settled value after the load change.	-230		230	mA	Section 7.2.6
<i>iPpsCLLoadStep</i>	Maximum Current set-point change while operating in <i>CL</i> mode.	-500		500	mA	Section 7.2.3.1
<i>iSafe0mA</i>	Maximum current a <i>Sink</i> is allowed to draw when <i>V_{BUS}</i> is driven to <i>vSafe0V</i> .			1.0	mA	Figure 7.29 Figure 7.30
<i>iSnkStdby</i>	Maximum current during voltage transition.			500	mA	Section 7.2.3
<i>iSnkSwapStdby</i>	Maximum current a <i>Sink</i> can draw during <i>Swap Standby</i> . Ideally this current is very near to 0 mA largely influenced by <i>Port</i> leakage current.			2.5	mA	Section 7.2.7

1) If more bypass capacitance than ***cSnkBulk*** max or ***cSnkBulkPd*** max is required in the device, then the device **shall** incorporate some form of *V_{BUS}* surge current limiting as described in [\[USB 3.2\]](#) Section 11.4.4.1.

Table 7.24 Sink Electrical Parameters (Continued)

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>pHubSusp</i>	Suspend power consumption for a <i>Hub</i> . 25mW + 25mW per downstream <i>Port</i> for up to 4 ports.			125	mW	Section 7.2.3
<i>pSnkSusp</i>	Suspend power consumption for a peripheral device.			25	mW	Section 7.2.3
<i>tNewSrc</i>	Maximum time allowed for an <i>Initial Sink</i> in <i>Swap Standby</i> to transition to <i>New Source</i> operation.			275	ms	Section 7.2.7 Table 7.18 Table 7.19
<i>tSnkFRSwap</i>	Time during a <i>Fast Role Swap</i> when the <i>New Sink</i> can draw no more than <i>iSnkStdby</i> .			200	μs	Section 7.1.13
<i>tSnkHardResetPrepare</i>	Time allotted for the <i>Sink</i> power electronics to prepare for a <i>Hard Reset</i> .			15	ms	Table 7.12
<i>tSnkNewPower</i>	Maximum transition time between power levels.			15	ms	Section 7.2.3
<i>tSnkRecover</i>	Time for the <i>Sink</i> to resume <i>USB Default Operation</i> .			150	ms	Table 7.20
<i>tSnkStdby</i>	Time to transition to <i>Sink Standby</i> from <i>Sink</i> .			15	ms	Section 7.2.3
<i>tSnkSwapStdby</i>	Maximum time for the <i>Sink</i> to transition to <i>Swap Standby</i> .			15	ms	Section 7.2.7
<i>vEprMax</i>	Highest voltage an <i>EPR Sink</i> is expected to tolerate			55	V	Section 7.2.9.2
<i>vSprMax</i>	Highest voltage an <i>SPR Sink</i> is expected to tolerate			24	V	Section 7.2.9.2
1)	If more bypass capacitance than <i>cSnkBulk</i> max or <i>cSnkBulkPd</i> max is required in the device, then the device Shall incorporate some form of <i>VBUS</i> surge current limiting as described in [USB 3.2] Section 11.4.4.1.					

7.4.3 Common Electrical Parameters

Electrical Parameters that are common to both the *Source* and the *Sink* that **shall** be followed are specified in [Table 7.25, "Common Source/Sink Electrical Parameters"](#).

Table 7.25 Common Source/Sink Electrical Parameters

Parameter	Description	MIN	TYP	MAX	UNITS	Reference
<i>tSafe0V</i>	Time to reach <i>vSafe0V</i> max.			650	ms	Section 7.1.5 Figure 7.10 Table 7.20 Table 7.21
<i>tSafe5V</i>	Time to reach <i>vSafe5V</i> max.			275	ms	Section 7.1.5 Figure 7.10 Table 7.20 Table 7.21
<i>tVCONNReapplied</i>	When the <i>UFP</i> is the <i>VCONN Source</i> : time from the last bit of the <i>GoodCRC</i> acknowledging the <i>PS_RDY</i> Message before reapplying <i>VCONN</i> . When the <i>DFP</i> is the <i>VCONN Source</i> : time from when <i>VCONN</i> drops below <i>vRaReconnect</i> .	10		20	ms	Figure 7.19 Figure 7.20
<i>tVCONNValid</i>	Time from <i>tVCONNReapplied</i> until <i>VCONN</i> is within <i>vVconnValid</i> (see [USB Type-C 2.4]). ¹	0		5	ms	Figure 7.19 Figure 7.20
<i>tVCONNZero</i>	Time from the last bit of the <i>GoodCRC</i> acknowledging the <i>Accept</i> Message in response to the <i>Data_Reset</i> Message until <i>VCONN</i> is below <i>vRaReconnect</i> (see [USB Type-C 2.4]).			125	ms	Figure 7.19 Figure 7.20
<i>vSafe0V</i>	Safe operating voltage at "zero volts".	0		0.8	V	Section 7.1.5
<i>vSafe5V</i>	Safe operating voltage at 5V. See [USB 2.0] and [USB 3.2] for allowable <i>VBUS</i> voltage range.	4.75		5.5	V	Section 7.1.5
1) <i>tVCONNStable</i> (See [USB Type-C 2.4]) still applies.						

8 Device Policy

8.1 Overview

This section describes the *Device Policy* and *Policy Engine* that implements it. For an overview of the architecture and how the *Device Policy Manager* fits into this architecture, please see [Section 2.6, "Architectural Overview"](#).

8.2 Device Policy Manager

The *Device Policy Manager* is responsible for managing the power used by one or more USB Power Delivery ports. In order to have sufficient knowledge to complete this task it needs relevant information about the device it resides in. Firstly, it has a priori knowledge of the device including the *Capabilities* of the power supply and the receptacles on each *Port* since these will for example have specific current ratings. It also has to know information from the *USB-C® Port Control* module regarding cable insertion, type and rating of cable etc. It also has to have information from the power supply about changes in its *Capabilities* as well as being able to request power supply changes. With all of this information the *Device Policy Manager* is able to provide up to date information regarding the *Capabilities* available to a specific *Port* and to manage the power resources within the device.

When working out the *Capabilities* for a given *Source Port* the *Device Policy Manager* will take into account firstly the current rating of the *Port's* receptacle and whether the inserted cable is PD or non-PD rated and if so, what is the capability of the plug. This will set an upper bound for the *Capabilities* which might be offered. After this the *Device Policy Manager* will consider the available power supply resources since this will bound which voltages and currents might be offered. Finally, the *Device Policy Manager* will consider what power is currently allocated to other ports, which power is in the Power Reserve and any other amendments to Policy from the *System Policy Manager*. The *Device Policy Manager* will offer a set of *Capabilities* within the bounds detailed above.

When selecting a capability for a given *Sink Port* the *Device Policy Manager* will look at the *Capabilities* offered by the *Source*. This will set an upper bound for the *Capabilities* which might be requested. The *Device Policy Manager* will also consider which *Capabilities* are required by the *Sink* in order to operate. If an appropriate match for voltage and Current can be found within the limits of the receptacle and cable, then this will be requested from the *Source*. If an appropriate match cannot be found then a request for an offered voltage and current will be made, along with an indication of a *Capabilities Mismatch*.

USB PD defines two types of power sources:

- Predefined voltage sources (*Fixed Supply*, *Variable Supply* and *Battery Supply*)
- Programmable voltage sources:
 - Programmable Power Supply (PPS)
 - *Adjustable Voltage Supply* (AVS)

The first are generally used for classic charging wherein the *Charger* electronics reside inside the *Sink*. The *Device Policy Manager* in the *Sink* requests a fixed voltage from the list of PDOs offered by the *Source* and which is converted internally to charge the *Sink's Battery* and/or power its function.

The second moves the *Charger* electronics that manage the voltage control outside the *Sink* and back into the *Source* itself. When in *SPR PPS Mode*, the *Device Policy Manager* in the *Sink* requests a specific voltage with a 20mV accuracy and sets a current limit. Unlike traditional USB where *Sinks* are responsible for limiting the current, they consume, the *SPR PPS Source* limits the current to what the *Sink* has requested. When operating in, the *Device Policy Manager* in the *Sink* requests a specific voltage with a 100mV accuracy and requests a maximum current it is allowed to draw.

Note: The AVS Sources unlike SPR PPS Sources do not support current limit mode. A *Sink* operating in is responsible not to draw more current than it requests.

The process to request power is the same for both types of power *Sources* although the actual format and contents of the request are slightly different. The primary operational differences are:

- A *Sink* that is using *SPR PPS* is required to periodically sent requests to let the *Source* know it is still alive and communicating. When this communication fails a *Hard Reset* results.
- A *Sink* operating in *SPR Mode* has no special timing requirements.
- A *Sink* operating in *EPR Mode* is required to periodically communicate with the *Source* to let it know it is still operational. If the communication fails, a *Hard Reset* results.

For *Dual-Role Power Ports* the *Device Policy Manager* manages the functionality of both a *Source* and a *Sink*. In addition, it is able to manage the *Power Role Swap* process between the two. In terms of power management this

could mean that a *Port* which is initially consuming power as a *Sink* is able to become a power resource as a *Source*. Conversely, *Attached Sources* might request that power be provided to them.

The functionality within the *Device Policy Manager* (and to a certain extent the *Policy Engine*) is scalable depending on the complexity of the device, including the number of different power supply *Capabilities* and the number of different features supported for example *System Policy Manager* interface or *Capabilities Mismatch*, and the number of ports being managed. Within these parameters it is possible to implement devices from very simple power supplies to more complex power supplies or devices such as *USB Hubs* or *Hard Drives*. Within multi-*Port* devices it is also permitted to have a combination of USB Power Delivery and non-USB Power Delivery ports which **Should** all be managed by the *Device Policy Manager*.

As noted in [Section 2.6, "Architectural Overview"](#) the logical architecture used in the PD specification will vary depending on the implementation. This means that different implementations of the *Device Policy Manager* might be relatively small or large depending on the complexity of the device, as indicated above. It is also possible to allocate different responsibilities between the *Policy Engine* and the *Device Policy Manager*, which will lead to different types of architectures and interfaces.

The *Device Policy Manager* is responsible for the following:

- Maintaining the *Local Policy* for the device.
- For a *Source*, monitoring the present *Capabilities* and triggering notifications of the change.
- For a *Sink*, evaluating and responding to *Capabilities* related requests from the *Policy Engine* for a given *Port*.
- Control of the *Source/Sink* in the device.
- Control of the USB-C[®] *Port Control* module for each *Port*.
- Interface to the *Policy Engine* for a given *Port*.

The *Device Policy Manager* is responsible for the following **Optional** features when implemented:

- Communications with the *System Policy* over USB.
- For *Sources* with multiple ports monitoring and balancing power requirements across these ports.
- Monitoring of batteries and AC power supplies.
- Managing Modes in its *Port Partner* and *Cable Plug(s)*.

8.2.1 Capabilities

The *Device Policy Manager* in a *Provider* **Shall** know the power supplies available in the device and their *Capabilities*. In addition, it **Shall** be aware of any other PD sources of power such as batteries and AC inputs. The available power sources and existing demands on the device **Shall** be taken into account when presenting *Capabilities* to a *Sink*.

The *Device Policy Manager* in a *Consumer* **Shall** know the requirements of the *Sink* and use this to evaluate the *Capabilities* offered by a *Source*. It **Shall** be aware of its own power sources e.g., Batteries or AC supplies where these have a bearing on its operation as a *Sink*.

The *Device Policy Manager* in a *Dual-Role Power Device* **Shall** combine the above *Capabilities* and **Shall** also be able to present the dual-role nature of the device to an *Attached* PD Capable device.

8.2.2 System Policy

A given PD Capable device might have no USB capability, or PD might have been added to a USB device in such a way that PD is not integrated with USB. In these two cases there **Shall** be no requirement for the *Device Policy Manager* to interact with the USB interface of the device. The following requirements **Shall** only apply to PD devices that expose PD functionality over USB.

The *Device Policy Manager* **shall** communicate over USB with the *System Policy Manager* according to the requirements detailed in [\[UCSI\]](#). Whenever requested the *Device Policy Manager* **shall** implement a *Local Policy* according to that requested by the *System Policy Manager*. For example, the *System Policy Manager* might request that a *Battery* powered Device temporarily stops charging so that there is sufficient power for an HDD to spin up.

Note: Due to timing constraints, a PD Capable device **shall** be able to respond autonomously to all time-critical PD related requests.

8.2.3 Control of Source/Sink

The *Device Policy Manager* for a *Provider* **shall** manage the power supply for each PD *Source Port* and **shall** know at any given time what the *Negotiated* power is. It **shall** request transitions of the supply and inform the *Policy Engine* whenever a transition completes.

The *Device Policy Manager* for a *Consumer* **shall** manage the *Sink* for each PD *Sink Port* and **shall** know at any given time what the *Negotiated* power is.

The *Device Policy Manager* for a *Dual-Role Power Device* **shall** manage the transition between *Source/Sink Power Roles* for each PD *Dual-Role Power Port* and **shall** know at any given time what *Power Role* the *Port* is in.

8.2.4 Cable Detection

8.2.4.1 Device Policy Manager in a Provider

The *Device Policy Manager* in the *Provider* **shall** control the USB-C[®] *Port Control* module and **shall** be able to use the USB-C[®] *Port Control* module to determine the *Attachment* status.

Note: It might be necessary for the *Device Policy Manager* to also initiate additional discovery using the *Discover Identity* Command in order to determine the full *Capabilities* of the cabling (see [Section 6.4.4.3.1, "Discover Identity"](#)).

8.2.4.2 Device Policy Manager in a Consumer

The *Device Policy Manager* in a *Consumer* controls the USB-C[®] *Port Control* module and **shall** be able to use the USB-C[®] *Port Control* module to determine the *Attachment* status.

8.2.4.3 Device Policy Manager in a Consumer/Provider

The *Device Policy Manager* in a *Consumer/Provider* inherits characteristics of *Consumers* and *Providers* and **shall** control the USB-C[®] *Port Control* module in order to support the *Dead Battery* back-powering case to determine the following for a given *Port*:

- *Attachment* of a USB Power Delivery *Provider/Consumer* which supports *Dead Battery* back-powering.
- Presence of *VBUS*.

8.2.4.4 Device Policy Manager in a Provider/Consumer

The *Device Policy Manager* in a *Provider/Consumer* inherits characteristics of *Consumers* and *Providers* and **may** control the USB-C[®] *Port Control* module in order to support the *Dead Battery* back-powering case to determine the following for a given *Port*:

- Presence of *VBUS*.

8.2.5 Managing Power Requirements

It is the responsibility of the *Device Policy Manager* in a *Provider* to be aware of the power requirements of all devices connected to its *Source Ports*. This includes being aware of any reserve power that might be required by devices in the future and ensuring that power is shared optimally amongst *Attached PD Capable* devices. This is a key function of the *Device Policy Manager*; whose implementation is critical to ensuring that all PD Capable devices get the power they require in a timely fashion in order to facilitate smooth operation. This is balanced by the fact that the *Device Policy Manager* is responsible for managing the sources of power that are, by definition, finite.

The *Consumer's Device Policy Manager* **Shall** ensure that it takes no more power than is required to perform its functions and when its requirements change, it **Should** make a new *Request*. The *Provider*, after satisfying the *Request*, **Should** reclaim any unused power to ensure that it can meet total power requirements of *Attached Sinks* on at least one *Port*.

Note: It is expected that a future design guide will provide additional guidance.

8.2.5.1 Managing the Power Reserve

There might be some products where a Device has certain functionality at one power level and a greater functionality at another, for example a Printer/Scanner that operates only as a printer with one power level and as a scanner if it can get more power. While the visibility of the linkage between power and functionality might only be apparent to the *USB Host*; the *Device Policy Manager* **Should** provide mechanisms to manage the power requirements of such Devices.

It is the *Device Policy Manager's* responsibility to allocate power and maintain a power reserve so as not to over-subscribe its available power resource. A Device with multiple ports such as a *Hub* **Shall** always attempt to meet the incremental demands of the *Port* requiring the highest incremental power from its power reserve.

8.2.5.2 Power Capability Mismatch

A *Capabilities Mismatch* occurs when a *Consumer* cannot obtain required power from a *Provider* (or the *Source* is not PD Capable) and the *Consumer* requires such *Capabilities* to operate. Different actions are taken by the *Device Policy Manager* and the *System Policy Manager* in this case.

8.2.5.2.1 Local device handling of mismatch

The *Consumer's Device Policy Manager* **Shall** cause a notification to be displayed to the end user that a power *Capabilities Mismatch* has occurred. Examples of such feedback can include:

- For a simple Device an LED **May** be used to indicate the failure. For example, during connection the LED could be solid amber. If the connection is successful, the LED could change to green. If the connection fails, it could be red or alternately blink amber.
- A more sophisticated Device with a user interface, e.g., a mobile device or monitor, **Should** provide notification through the user interface on the Device.

The *Provider's Device Policy Manager* **May** cause a notification to be displayed to the user of the power *Capabilities Mismatch*.

Because the *Capabilities Mismatch* might not cause operational failure, the *Provider's Device Policy Manager* **Should Not** display a notification to the user if the power offered to the *Sink* meets or exceeds the **SPR Sink Minimum PDP/EPR Sink Minimum PDP** Advertised in the *Sink Capabilities Extended* Message (see [Section 6.5.13, "Sink Capabilities Extended Message"](#)). If a notification is displayed, it **Should Not** be shown as an error unless the power offered to the *Sink* is less than the **SPR Sink Minimum PDP/EPR Sink Minimum PDP** Advertised in the *Sink Capabilities Extended* Message.

8.2.5.2.2 Device Policy Manager Communication with System Policy

In a USB Power Delivery aware system with an active *System Policy Manager* (see [Section 8.2.2, "System Policy"](#)), the *Device Policy Manager* **Shall** notify the *System Policy Manager* of the mismatch. This information **Shall** be passed back to the *System Policy Manager* using the mechanisms described in [\[UCSI\]](#). The *System Policy Manager* **Should**

ensure that the user is informed of the condition. When another *Port* in the system could satisfy the *Consumer's* power requirements the user **Should** be directed to move the *Device* to the alternate *Port*.

In order to identify a more suitable *Source Port* for the *Consumer* the *System Policy Manager* **Shall** communicate with the *Device Policy Manager* in order to determine the *Consumer's* requirements. The *Device Policy Manager* **Shall** use a **Get_Sink_Cap** Message (see [Section 6.3.8, "Get_Sink_Cap Message"](#)) to discover which power levels can be utilized by the *Consumer*.

8.2.6 Use of "Unconstrained Power" bit with Batteries and AC supplies

The *Device Policy Manager* in a *Provider* or *Consumer* **May** monitor the status of any variable sources of power that could have an impact on its *Capabilities* as a *Source* such as Batteries and AC supplies and reflect this in the "Unconstrained Power" bit (see [Section 6.4.1.2.1.3, "Unconstrained Power"](#) and [Section 6.4.1.3.1.3, "Unconstrained Power"](#)) provided as part of the **Source_Capabilities** or **Sink_Capabilities** Message (see [Section 6.4.1, "Capabilities Message"](#)). When monitored, and a USB interface is supported, the External Power status (see [\[UCSI\]](#)) and the Battery state (see [Section 9.4.1, "GetBatteryStatus"](#)) **Shall** also be reported to the *System Policy Manager* using the USB interface.

8.2.6.1 AC Supplies

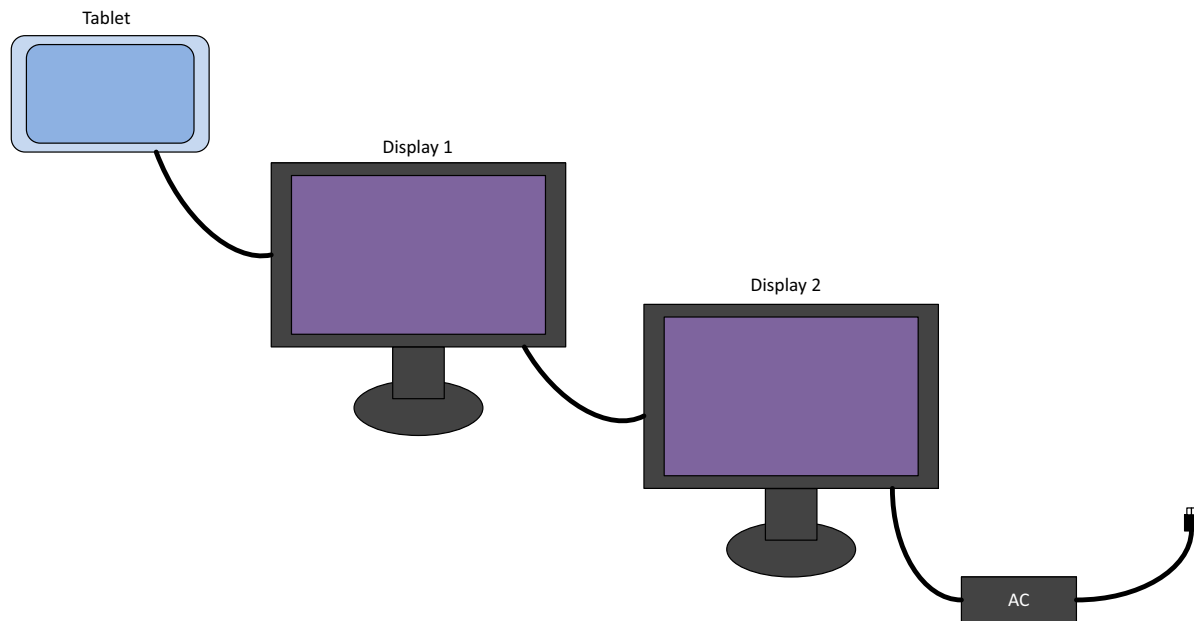
The Unconstrained Power bit provided by *Sources* and *Sinks* (see [Section 6.4.1.2.1.3, "Unconstrained Power"](#) and [Section 6.4.1.3.1.3, "Unconstrained Power"](#)) notifies a connected device that it is acceptable to use the *Advertised* power for charging as well as for what is needed for normal operation. A device that sets the Unconstrained Power bit has either an external source of power that is sufficient to adequately power the system while charging external devices or expects to charge external devices as a primary state of function (such as a battery pack).

In the case of the external power source, the power can either be from an *AC Supply* directly connected to the device or from an *AC Supply* connected to an *Attached* device, which is also getting unconstrained power from its power supply. The Unconstrained Power bit is in this way communicated through a PD system indicating that the origin of the power is from a single or multiple AC supplies, from a battery bank, or similar:

- If the "Unconstrained Power" bit is set, then that power is originally sourced from an *AC Supply*.
- Devices capable of consuming on multiple ports can only claim that they have "Unconstrained Power" for the power *Advertised* as a *Provider Port* if there is unconstrained power beyond that needed for normal operation coming from external supplies, (e.g., multiple AC supplies).
- This concept applies as the power is routed through multiple *Provider* and *Consumer* tiers, so, as an example. Power provided out of a monitor that is connected to a monitor that gets power from an *AC Supply*, will claim it has "Unconstrained Power" even though it is not directly connected to the *AC Supply*.

An example use case is a Tablet computer that is used with two USB A/V displays that are daisy chained (see [Figure 8.1, "Example of daisy chained displays"](#)). The tablet and 1st display are not externally powered, (meaning, they have no source of power outside of USB PD). The 2nd display has an external supply *Attached* which could either be a USB PD based supply or some other form of external supply. When the displays are connected as shown, the power adapter *Attached* to the 2nd display is able to power both the 1st display and the tablet. In this case the 2nd display will indicate the presence of a sufficiently sized *Charger* to the 1st display, by setting its "Unconstrained Power" bit. The 1st display will then in turn assess and indicate the presence of the extra power to the tablet by setting its "Unconstrained Power" bit. Power is transmitted through the system to all devices, provided that there is sufficient power available from the external supply.

Figure 8.1 Example of daisy chained displays



Another example use case is a laptop computer that is *Attached* to both an external supply and a Tablet computer. In this situation, if the external supply is large enough to power the laptop in its normal state as well as charge an external device, the laptop would set its "Unconstrained Power" bit and the tablet will allow itself to charge at its peak rate. If the external supply is small, however, and would not prevent the laptop from discharging if maximal power is drawn by the external device, the laptop would not set its "Unconstrained Power" bit, and the tablet can choose to draw less than what is offered. This amount could be just enough to prevent the tablet from discharging, or none at all. Alternatively, if the tablet determines that the laptop has significantly larger battery with more charge than the tablet has, the tablet can still choose to charge itself, although possibly not at the maximal rate.

In this way, *Sinks* that do not receive the **Unconstrained Power** bit from the connected *Source* can still choose to charge their batteries, or charge at a reduced rate, if their policy determines that the impact to the *Source* is minimal -- such as in the case of a phone with a small battery charging from a laptop with a large battery. These policies can be decided via further USB PD communication.

8.2.6.2 Battery Supplies

When monitored, and a USB interface is supported, the *Battery* state **Shall** be reported to the *System Policy Manager* using the USB interface.

If the device is *Battery*-powered but is in a state that is primarily for charging external devices, the device is considered to be an unconstrained source of power and thus **Should** set the "Unconstrained Power" bit.

A simplified algorithm is detailed below to ensure that *Battery* powered devices will get charge from non-*Battery* powered devices when possible, and also to ensure that devices do not constantly *Power Role Swap* back and forth.

When two devices are connected that do not have Unconstrained Power, they **Should** define their own policies so as to prevent constant *Power Role Swapping*.

This algorithm uses the "Unconstrained Power" bit (see [Section 6.4.1.2.1.3, "Unconstrained Power"](#) and [Section 6.4.1.3.1.3, "Unconstrained Power"](#)), thus the decisions are based on the availability and sufficiency of an external supply, not the full *Capabilities* of a system or device or product.

Recommendations:

- *Provider/Consumers* using large external sources ("Unconstrained Power" bit set) **Should** always deny *Power Role Swap* requests from *Consumer/Providers* not using external sources ("Unconstrained Power" bit cleared).

- *Provider/Consumers* not using large external sources ("Unconstrained Powered" bit cleared) **Should** always accept a *Power Role Swap* request from a *Consumer/Provider* using large external power sources ("Unconstrained Power" bit set) unless the requester is not able to provide the requirements of the present *Provider/Consumer*.

8.2.7 Interface to the Policy Engine

The *Device Policy Manager* **Shall** maintain an interface to the *Policy Engine* for each *Port* in the device.

8.2.7.1 Device Policy Manager in a Provider

The *Device Policy Manager* in a *Provider* **Shall** also provide the following functions to the *Policy Engine*:

- Inform the *Policy Engine* of changes in cable/ device *Attachment* status for a given cable.
- Inform the *Policy Engine* whenever the *Source Capabilities* available for a *Port* change.
- Evaluate requests from an *Attached Consumer* and provide responses to the *Policy Engine*.
- Respond to requests for power supply transitions from the *Policy Engine*.
- Indication to *Policy Engine* when power supply transitions are complete.
- Maintain a power reserve for devices operating on a *Port* at less than maximum power.

8.2.7.2 Device Policy Manager in a Consumer

The *Device Policy Manager* in a *Consumer* **Shall** also provide the following functions to the *Policy Engine*:

- Inform the *Policy Engine* of changes in cable/device *Attachment* status.
- Inform the *Policy Engine* whenever the power requirements for a *Port* change.
- Evaluate *Source Capabilities* and provide suitable responses:
 - Request from offered *Capabilities*.
 - Indicate whether additional power is required.
- Respond to requests for *Sink* transitions from the *Policy Engine*.

8.2.7.3 Device Policy Manager in a Dual-Role Power Device

The *Device Policy Manager* in a *Dual-Role Power Device* **Shall** provide the following functions to the *Policy Engine*:

- *Provider Device Policy Manager*
- *Consumer Device Policy Manager*
- Interface for the *Policy Engine* to request power supply transitions from *Source* to *Sink* and vice versa.
- Indications to *Policy Engine* during *Power Role Swap* transitions.

8.2.7.4 Device Policy Manager in a Dual-Role Power Device Dead Battery handling

The *Device Policy Manager* in a *Dual-Role Power Device* with a *Dead Battery* **Should**:

- Switch *Ports* to *Sink*-only or *Sink DFP* operation to obtain power from the next *Attached Source*.
- Use *V_{BUS}* from the *Attached Source* to power the USB Power Delivery communications as well as charging to enable the *Negotiation* of higher input power.

8.3 Policy Engine

8.3.1 Introduction

There is one *Policy Engine* instance per *Port* that interacts with the *Device Policy Manager* in order to implement the present *Local Policy* for that particular *Port*. This section includes:

- *AMSs* for various operations.
- State diagrams covering operation of *Sources*, *Sinks* and *Cable Plugs*.

8.3.2 Atomic Message Sequence Diagrams

8.3.2.1 Introduction

The *Policy Engine* drives the *Atomic Message Sequences* (AMS) and responses based on both the expected AMSs and the present *Local Policy*.

An AMS **Shall** be defined as a *Message* sequence that starts and/or ends in either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** states (see [Section 8.3.3.2, "Policy Engine Source Port State Diagram"](#), [Section 8.3.3.3, "Policy Engine Sink Port State Diagram"](#) and [Section 8.3.3.25, "Cable Plug Specific State Diagrams"](#)).

In addition, the *Cable Plug* discovery sequence specified in [Section 8.3.3.25.3, "Source Startup Structured VDM Discover Identity of a Cable Plug State Diagram"](#) **Shall** be defined as an AMS.

The *Source* and *Sink* indicate to the *Protocol Layer* when an AMS starts and ends on entry to/exit from **PE_SRC_Ready** or **PE_SNK_Ready** (see [Section 8.3.3.2, "Policy Engine Source Port State Diagram"](#) and [Section 8.3.3.3, "Policy Engine Sink Port State Diagram"](#)).

An AMS **Shall** be considered to have been started by the *Initiator* when the protocol engine signals the *Policy Engine* that transmission is a success (the **GoodCRC** Message has been received in response to the initial *Message*). For the receiving *Port* the AMS **Shall** be considered to have started when the initial *Message* has arrived.

An AMS **Shall** be considered to have ended:

- When the *Protocol Layer* signals the *Policy Engine* that transmission of the final *Message* in the AMS is a success and for the opposite *Port* when the final *Message* has been received.
- A **Soft_Reset** Message, **Hard_Reset** Signaling for SOP' or SOP'' or **Cable_Reset** Signaling has been sent or received.

[Section 8.3.2.1.3, "Atomic Message Sequences"](#) gives details of these AMS's.

This section contains sequence diagrams that highlight some of the more interesting transactions. It is by no means a complete summary of all possible combinations but is **Informative** in nature.

8.3.2.1.1 Basic Message Exchange

Figure 8.2, "Basic Message Exchange (Successful)" below illustrates how a *Message* is sent. Table 8.1, "Basic Message Flow" details the steps in the flow. Note that the sender might be either a *Source* or *Sink* while the receiver might be either a *Sink* or *Source*. The basic *Message* sequence is the same. It starts when the *Message Sender's Protocol Layer* at the behest of its *Policy Engine* forms a *Message* that it passes to the *PHY Layer*.

Figure 8.2 Basic Message Exchange (Successful)

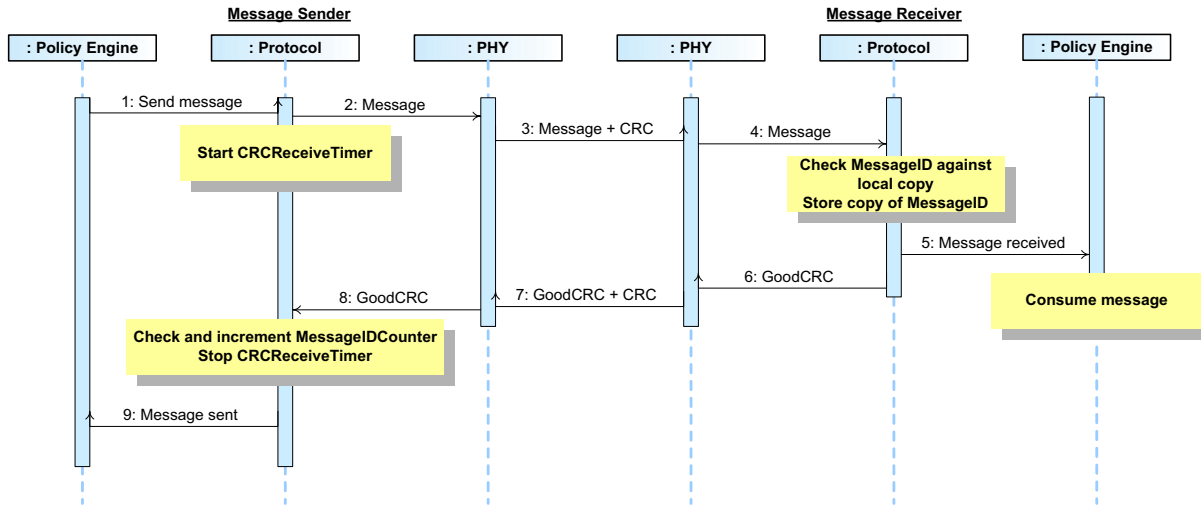


Table 8.1 Basic Message Flow

Step	Message Sender	Message Receiver
1	Policy Engine directs Protocol Layer to send a Message.	
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends a CRC and sends the Message. Starts CRCReceiveTimer .	PHY Layer receives the Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. Protocol Layer forwards the received Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it to the PHY Layer.
7	PHY Layer receives the Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer. Protocol Layer checks and increments the MessageIDCounter and stops CRCReceiveTimer .	
9	Protocol Layer informs the Policy Engine that the Message was successfully sent.	

8.3.2.1.2 Errors in Basic Message flow

There are various points during the *Message* flow where failures in communication or other issues can occur. [Figure 8.3, "Basic Message flow indicating possible errors"](#) is an annotated version of [Figure 8.2, "Basic Message Exchange \(Successful\)"](#) indicating at which point issues can occur. [Table 8.2, "Potential issues in Basic Message Flow"](#) details the steps in the flow.

Figure 8.3 Basic Message flow indicating possible errors

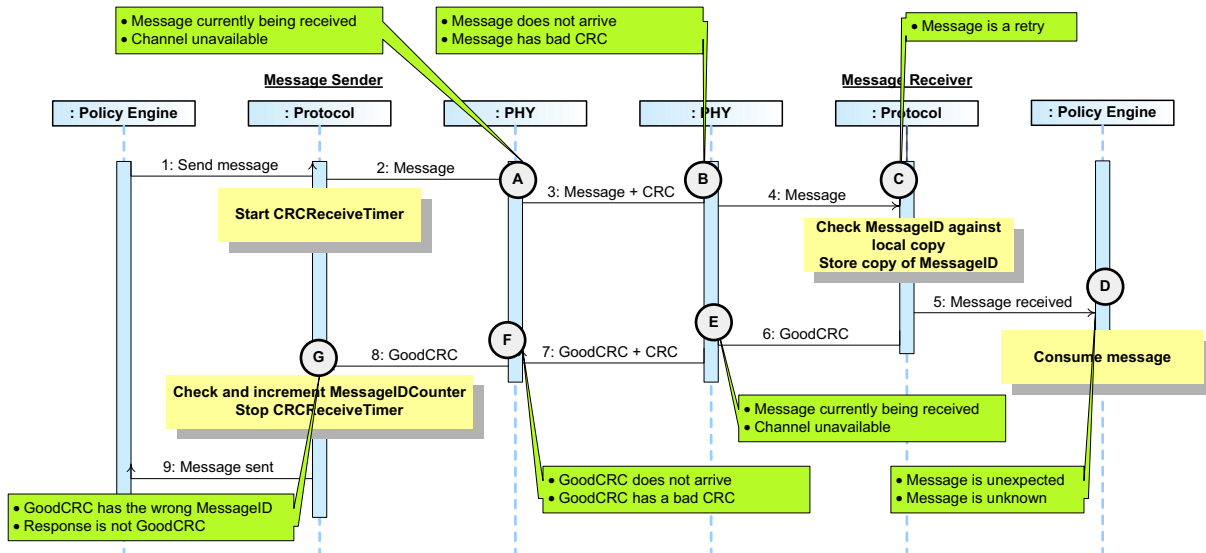


Table 8.2 Potential issues in Basic Message Flow

Point	Possible issues
A	<ol style="list-style-type: none"> 1) There is an incoming <i>Message</i> on the channel meaning that the <i>PHY Layer</i> is unable to send. In this case the outgoing <i>Message</i> is removed from the queue and the incoming <i>Message</i> processed. 2) Due to some sort of noise on the line it is not possible to transmit. In this case the outgoing <i>Message</i> is Discarded by the <i>PHY Layer</i>. Retransmission is via the <i>Protocol Layer</i>'s normal mechanism.
B	<ol style="list-style-type: none"> 1) <i>Message</i> does not arrive at the <i>PHY Layer</i> due to noise on the channel. 2) <i>Message</i> arrives but has been corrupted and has a bad <i>CRC</i>. <p>There is no <i>Message</i> to pass up to the <i>Protocol Layer</i> on the receiver which means a <i>GoodCRC Message</i> is not sent. This leads to a <i>CRCReceiveTimer</i> timeout in the <i>Message Sender</i>.</p>
C	<ol style="list-style-type: none"> 1) <i>MessageID</i> of received <i>Message</i> matches stored <i>MessageID</i> so this is a retry. <i>Message</i> is not passed up to the <i>Policy Engine</i>.
D	<ol style="list-style-type: none"> 1) <i>Policy Engine</i> receives a known <i>Message</i> that it was not expecting. 2) <i>Policy Engine</i> receives an <i>Unrecognized Message</i>. <p>These cases are errors in the protocol which could lead to the generation of a <i>Soft_Reset Message</i>.</p>
E	Same as point A but at the <i>Message Receiver</i> side.

Table 8.2 Potential issues in Basic Message Flow

Point	Possible issues
F	<p>1) GoodCRC Message response does not arrive at the <i>Message Sender</i> side due to the noise on the channel.</p> <p>2) GoodCRC Message response arrives but has a bad CRC.</p> <p>A GoodCRC Message is not received by the <i>Message Sender's Protocol Layer</i>. This leads to a CRCReceiveTimer timeout in the <i>Message Sender</i>.</p>
G	<p>1) GoodCRC Message is received but does contain the same MessageID as the transmitted <i>Message</i>.</p> <p>2) A <i>Message</i> is received but it is not a GoodCRC Message (similar case to that of an unexpected or unknown <i>Message</i> but this time detected in the <i>Protocol Layer</i>).</p> <p>Both of these issues indicate errors in receiving an expected GoodCRC Message which will lead to a CRCReceiveTimer timeout in the <i>Protocol Layer</i> and a subsequent retry (except for communications with <i>Cable Plugs</i>).</p>

Figure 8.4, "Basic Message Flow with Bad followed by a Retry" illustrates one of these cases; the basic *Message* flow with a retry due to a bad CRC at the *Message Receiver*. It starts when the *Message Sender's Protocol Layer* at the behest of its *Policy Engine* forms a *Message* that it passes to the *PHY Layer*. The *Protocol Layer* is responsible for retries on a "'n' strikes and you are out" basis (**nRetryCount**). Table 8.3, "Basic Message Flow with CRC failure" details the steps in the flow.

Figure 8.4 Basic Message Flow with Bad followed by a Retry

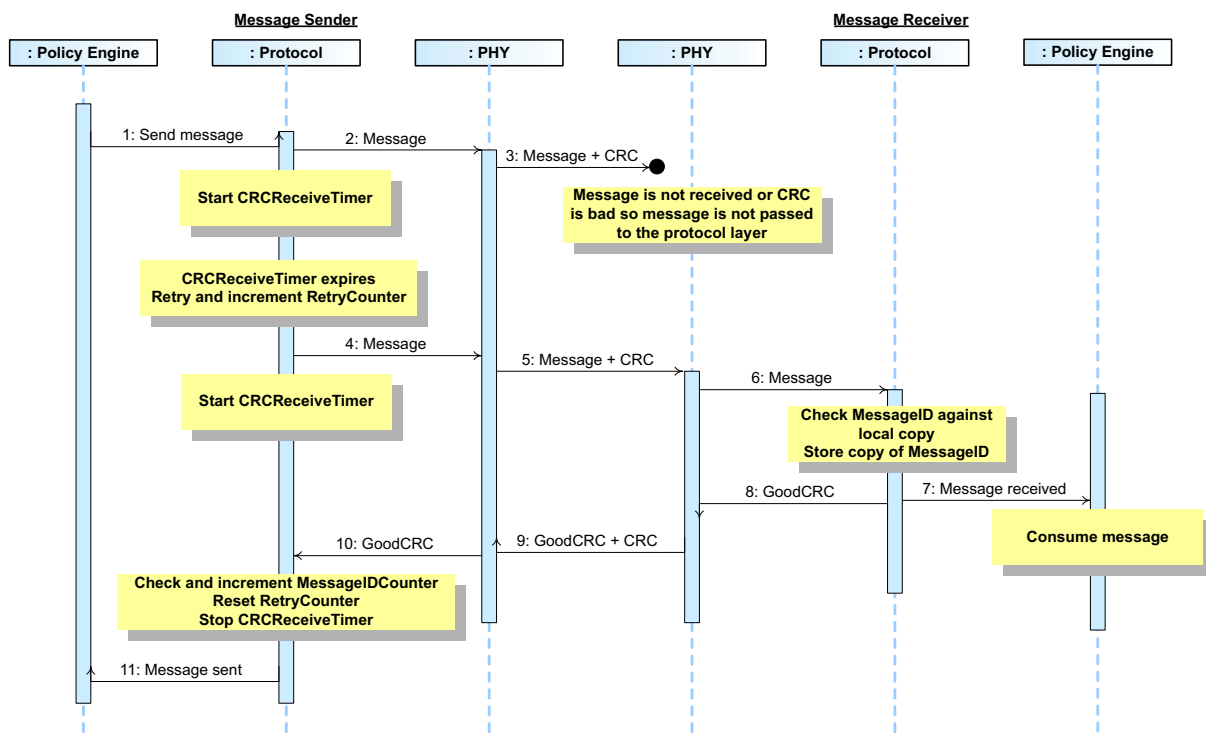


Table 8.3 Basic Message Flow with CRC failure

Step	Message Sender	Message Receiver
1	<i>Policy Engine directs Protocol Layer to send a Message.</i>	
2	<i>Protocol Layer creates the Message and passes to PHY Layer.</i>	
3	<i>PHY Layer appends a CRC and sends the Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives no Message or a Message with an incorrect CRC. Nothing is passed to Protocol Layer.</i>
4	<i>Since no response is received, the CRCReceiveTimer will expire and trigger the first retry by the Protocol Layer. The RetryCounter is incremented. Protocol Layer passes the Message to the PHY Layer.</i>	
5	<i>PHY Layer appends a CRC and sends the Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Message and checks the CRC to verify the Message.</i>
6		<i>PHY Layer removes the CRC and forwards the Message to the Protocol Layer.</i>
7		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. Protocol Layer forwards the received Message information to the Policy Engine that consumes it.</i>
8		<i>Protocol Layer generates a GoodCRC Message and passes it to the PHY Layer.</i>
9	<i>PHY Layer receives the Message and checks the CRC to verify the Message.</i>	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>
10	<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>	
11	<i>Protocol Layer verifies the MessageID, stops CRCReceiveTimer and resets the RetryCounter. Protocol Layer informs the Policy Engine that the Message was successfully sent.</i>	

8.3.2.1.3 Atomic Message Sequences

The types of *Atomic Message Sequences* (AMS) are listed in [Table 8.4, "Atomic Message Sequences"](#). The following tables list sequences of either *Messages* or combinations of *Messages* and one or more embedded AMSes which are *Non-interruptible*. Where there is an embedded AMS the entire *Message* sequence is treated as an AMS and the R_p value used for *Collision Avoidance* (see [Section 5.7, "Collision Avoidance"](#)) **Shall** only be changed on leaving or entering the ready state at the beginning or end of the entire *Message* sequence, and not at the start or end of the embedded AMS.

Note: An AMS is has not started until the first *Message* in the sequence has been successfully sent (i.e., a **GoodCRC** *Message* has been received acknowledging the *Message*).

[Table 8.31, "AMS: Hard Reset"](#) details a *Hard Reset* (which is *Signaling* not an AMS) followed by an *SPR Contract Negotiation AMS* which **Shall** be treated as *Non-interruptible*.

Table 8.4 Atomic Message Sequences

Type of AMS	Table Reference	Section Reference
Power Negotiation (SPR)	Table 8.5, "AMS: Power Negotiation (SPR)"	Section 8.3.2.2.1
Power Negotiation (EPR)	Table 8.6, "AMS: Power Negotiation (EPR)"	Section 8.3.2.2.2
Unsupported Message	Table 8.7, "AMS: Unsupported Message"	Section 8.3.2.3
Soft Reset	Table 8.8, "AMS: Soft Reset"	Section 8.3.2.4
Data Reset	Table 8.9, "AMS: Data Reset"	Section 8.3.2.5
Hard Reset	Table 8.31, "AMS: Hard Reset"	Section 8.3.2.6
Power Role Swap	Table 8.10, "AMS: Power Role Swap"	Section 8.3.2.7
Fast Role Swap	Table 8.11, "AMS: Fast Role Swap"	Section 8.3.2.8
Data Role Swap	Table 8.12, "AMS: Data Role Swap"	Section 8.3.2.9
VCONN Swap	Table 8.13, "AMS: VCONN Swap"	Section 8.3.2.10
Alert	Table 8.14, "AMS: Alert"	Section 8.3.2.11.1
Status	Table 8.15, "AMS: Status"	Section 8.3.2.11.2
Source Capabilities/ Sink Capabilities (SPR)	Table 8.16, "AMS: Source/Sink Capabilities (SPR)"	Section 8.3.2.11.3.1
Source Capabilities/ Sink Capabilities (EPR)	Table 8.17, "AMS: Source/Sink Capabilities (EPR)"	Section 8.3.2.11.3.2
Extended Capabilities	Table 8.18, "AMS: Extended Capabilities"	Section 8.3.2.11.4
Battery Capabilities and Status	Table 8.19, "AMS: Battery Capabilities"	Section 8.3.2.11.5
Manufacturer Information	Table 8.20, "AMS: Manufacturer Information"	Section 8.3.2.11.6
Country Codes	Table 8.21, "AMS: Country Codes"	Section 8.3.2.11.7
Country Information	Table 8.22, "AMS: Country Information"	Section 8.3.2.11.8
Revision Information	Table 8.23, "AMS: Revision Information"	Section 8.3.2.11.9
Source Information	Table 8.24, "AMS: Source Information"	Section 8.3.2.11.10
Security	Table 8.25, "AMS: Security"	Section 8.3.2.12
Firmware Update	Table 8.26, "AMS: Firmware Update"	Section 8.3.2.13
Structured VDM	Table 8.27, "AMS: Structured VDM"	Section 8.3.2.14
Built-In Self-Test (BIST)	Table 8.28, "AMS: Built-In Self-Test (BIST)"	Section 8.3.2.15
Enter USB	Table 8.29, "AMS: Enter USB"	Section 8.3.2.16
Unstructured VDM	Table 8.30, "AMS: Unstructured VDM"	Section 8.3.2.17

Table 8.5 AMS: Power Negotiation (SPR)

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
SPR Explicit Contract Negotiation (Accept)	1. <i>Source_Capabilities</i> Message 2. <i>Request</i> Message 3. <i>Accept</i> Message 4. <i>PS_RDY</i> Message	Started by Source, SPR Mode	Section 8.3.2.2.1.1.1	Section 8.3.3.2 , Section 8.3.3.3
SPR Explicit Contract Negotiation (Reject)	1. <i>Source_Capabilities</i> Message 2. <i>Request</i> Message 3. <i>Reject</i> Message		Section 8.3.2.2.1.1.2	
SPR Explicit Contract Negotiation (Wait)	1. <i>Source_Capabilities</i> Message 2. <i>Request</i> Message 3. <i>Wait</i> Message		Section 8.3.2.2.1.1.3	
SPR PPS Keep Alive	1. <i>Request</i> Message 2. <i>Accept</i> Message 3. <i>PS_RDY</i> Message	Started by Sink, SPR Mode	Section 8.3.2.2.1.2	Section 8.3.3.3
SPR Sink Makes Request (Accept)	1. <i>Request</i> Message 2. <i>Accept</i> Message 3. <i>PS_RDY</i> Message		Section 8.3.2.2.1.3.1	Section 8.3.3.2 , Section 8.3.3.3
SPR Sink Makes Request (Reject)	1. <i>Request</i> Message 2. <i>Reject</i> Message		Section 8.3.2.2.1.3.2	
SPR Sink Makes Request (Wait)	1. <i>Request</i> Message 2. <i>Wait</i> Message		Section 8.3.2.2.1.3.3	

Table 8.6 AMS: Power Negotiation (EPR)

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Entering <i>EPR Mode</i> (Success)	<ol style="list-style-type: none"> 1. <i>EPR_Mode</i> (Enter) <i>Message</i> 2. <i>EPR_Mode</i> (Enter Acknowledge) <i>Message</i> 3. <i>VCONN Source</i> Swap, initiated by non- <i>VCONN Source</i> (Accept) AMS (<i>Optional</i>). 4. <i>Initiator to Responder</i> Discover Identity (ACK) AMS (<i>Optional</i> for Sources with captive cables) 5. <i>EPR_Mode</i> (Enter Succeeded) <i>Message</i> 6. EPR Explicit Contract Negotiation AMS 	Started by Sink, <i>SPR Mode</i>	Section 8.3.2.2.2.1 , Section 8.3.2.10.1 , Section 8.3.2.10.2 , Section 8.3.2.12.3 , Section 8.3.2.2.2.4	Section 8.3.3.25.1 , Section 8.3.3.25.2 , Section 8.3.3.19 , Section 8.3.3.20.1 , Section 8.3.3.21.1 , Section 8.3.3.2 , Section 8.3.3.3
Entering <i>EPR Mode</i> (Failure due to non- <i>EPR Cable</i>)	<ol style="list-style-type: none"> 1. <i>EPR_Mode</i> (Enter) <i>Message</i> 2. <i>EPR_Mode</i> (Enter Acknowledge) <i>Message</i> 3. <i>VCONN Source</i> Swap, initiated by non- <i>VCONN Source</i> (Accept) AMS(<i>Optional</i>). 4. <i>Initiator to Responder</i> Discover Identity (ACK) AMS (<i>Optional</i> for Sources with captive cables) 5. <i>EPR_Mode</i> (Enter Failed) <i>Message</i> 	Started by Sink, <i>SPR Mode</i>	Section 8.3.2.2.2.2 , Section 8.3.2.10.1 , Section 8.3.2.10.2 , Section 8.3.2.12.3	Section 8.3.3.25.1 , Section 8.3.3.25.2 , Section 8.3.3.19 , Section 8.3.3.20.1 , Section 8.3.3.21.1
Entering <i>EPR Mode</i> (Failure of <i>VCONN Swap</i>)	<ol style="list-style-type: none"> 1. <i>EPR_Mode</i> (Enter) <i>Message</i>. 2. <i>EPR_Mode</i> (Enter Acknowledge) <i>Message</i>. 3. <i>VCONN Source</i> Swap, initiated by non- <i>VCONN Source</i> (Reject) AMS(<i>Optional</i>). 4. <i>EPR_Mode</i> (Enter Failed) <i>Message</i> 	Started by Sink, <i>SPR Mode</i>	Section 8.3.2.2.2.3 , Section 8.3.2.10.1 , Section 8.3.2.10.2	Section 8.3.3.25.1 , Section 8.3.3.25.2 , Section 8.3.3.19

Table 8.6 AMS: Power Negotiation (EPR)

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
EPR Explicit Contract Negotiation (Accept)	1. EPR_Source_Capabilities Message 2. EPR_Request Message 3. Accept Message 4. PS_RDY Message	Started by Source, EPR Mode	Section 8.3.2.2.2.1	Section 8.3.3.2 , Section 8.3.3.3
EPR Explicit Contract Negotiation (Reject)	1. EPR_Source_Capabilities Message 2. EPR_Request Message 3. Reject Message		Section 8.3.2.2.2.2	
EPR Explicit Contract Negotiation (Wait)	1. EPR_Source_Capabilities Message 2. EPR_Request Message 3. Wait Message		Section 8.3.2.2.2.3	
EPR Keep Alive	1. EPR_KeepAlive Message 2. EPR_KeepAlive_Ack Message	Started by Sink, EPR Mode	Section 8.3.2.2.2.3	
Exiting EPR Mode (Sink Initiated)	1. EPR_Mode (Exit) Message 2. SPR Explicit Contract Negotiation AMS	Started by Sink, EPR Mode	Section 8.3.2.2.2.4.1 , Section 8.3.2.2.1.1	Section 8.3.3.25.3 , Section 8.3.3.25.4 , Section 8.3.3.2 , Section 8.3.3.3
Exiting EPR Mode (Source Initiated)	1. EPR_Mode (Exit) Message 2. SPR Explicit Contract Negotiation AMS	Started by Source, EPR Mode	Section 8.3.2.2.2.4.2 , Section 8.3.2.2.1.1	
EPR Sink Makes Request (Accept)	1. EPR_Request Message 2. Accept Message 3. PS_RDY Message	Started by Sink, EPR Mode	Section 8.3.2.2.2.5.1	Section 8.3.3.2 , Section 8.3.3.3
EPR Sink Makes Request (Reject)	1. EPR_Request Message 2. Reject Message	Started by Sink, EPR Mode	Section 8.3.2.2.2.5.2	
EPR Sink Makes Request (Wait)	1. EPR_Request Message 2. Wait Message	Started by Sink, EPR Mode	Section 8.3.2.2.2.5.3	

8.3.2.1.3.3

AMS: Unsupported Message

Table 8.7 AMS: Unsupported Message

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
<i>Unsupported Message</i>	<ol style="list-style-type: none"> Any <i>Message</i> which is not supported by the <i>Source</i> or <i>Sink</i> Not_Supported Message 	Started by <i>Source</i> or <i>Sink</i>	Section 8.3.2.3	Section 8.3.3.6.2

8.3.2.1.3.4

AMS: Soft Reset

Table 8.8 AMS: Soft Reset

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
<i>Soft Reset</i>	<ol style="list-style-type: none"> Soft_Reset Message Accept Message In <i>SPR Mode</i>: SPR Explicit Contract Negotiation AMS or in <i>EPR Mode</i>: EPR Explicit Contract Negotiation AMS. 	Started by <i>Source</i> or <i>Sink</i>	Section 8.3.2.4 , Section 8.3.2.2.1.1 , Section 8.3.2.2.1.1 , Section 8.3.2.2.2	Section 8.3.3.4.1 , Section 8.3.3.4.2 , Section 8.3.3.25.2.1 , Section 8.3.3.25.2.3 , Section 8.3.3.25.2.4 , Section 8.3.3.2 , Section 8.3.3.3

8.3.2.1.3.5

AMS: Data Reset

Table 8.9 AMS: Data Reset

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
<i>DFP Initiated Data Reset</i> where the <i>DFP</i> is the <i>VCONN Source</i>	<ol style="list-style-type: none"> Data_Reset Message Accept Message Data_Reset_Complete Message 	Started by <i>DFP</i>	Section 8.3.2.5.1	Section 8.3.3.5.1 , Section 8.3.3.5.2
<i>DFP Receives Data Reset</i> where the <i>DFP</i> is the <i>VCONN Source</i>	<ol style="list-style-type: none"> Data_Reset Message Accept Message Data_Reset_Complete Message 	Started by <i>UFP</i>	Section 8.3.2.5.2	
<i>DFP Initiated Data Reset</i> where the <i>UFP</i> is the <i>VCONN Source</i>	<ol style="list-style-type: none"> Data_Reset Message Accept Message PS_RDY Message Data_Reset_Complete Message 	Started by <i>DFP</i>	Section 8.3.2.5.3	
<i>DFP Receives Data Reset</i> where the <i>UFP</i> is the <i>VCONN Source</i>	<ol style="list-style-type: none"> Data_Reset Message Accept Message PS_RDY Message Data_Reset_Complete Message 	Started by <i>UFP</i>	Section 8.3.2.5.4	

8.3.2.1.3.6

AMS: Power Role Swap

Table 8.10 AMS: Power Role Swap

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source Initiated Power Role Swap (Accept)	1. PR_Swap Message 2. Accept Message 3. PS_RDY Message 4. PS_RDY Message 5. SPR Explicit Contract Negotiation AMS	Started by Source	Section 8.3.2.7.1.1 , Section 8.3.2.2.1.1	Section 8.3.3.19.3 , Section 8.3.3.19.4 , Section 8.3.3.2 , Section 8.3.3.3
Source Initiated Power Role Swap (Reject)	1. PR_Swap Message 2. Reject Message		Section 8.3.2.7.1.2	
Source Initiated Power Role Swap (Wait)	1. PR_Swap Message 2. Wait Message		Section 8.3.2.7.1.1	
Sink Initiated Power Role Swap (Accept)	1. PR_Swap Message 2. Accept Message 3. PS_RDY Message 4. PS_RDY Message 5. SPR Explicit Contract Negotiation AMS	Started by Sink	Section 8.3.2.7.2.1 , Section 8.3.2.2.1.1	
Sink Initiated Power Role Swap (Reject)	1. PR_Swap Message 2. Reject Message		Section 8.3.2.7.2.2	
Sink Initiated Power Role Swap (Wait)	1. PR_Swap Message 2. Wait Message		Section 8.3.2.7.2.3	

8.3.2.1.3.7

AMS: Fast Role Swap

Table 8.11 AMS: Fast Role Swap

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Fast Role Swap	1. FR_Swap Message 2. Accept Message 3. PS_RDY Message 4. PS_RDY Message 5. SPR Explicit Contract Negotiation AMS	Started by Sink	Section 8.3.2.8 , Section 8.3.2.2.1.1	Section 8.3.3.2 , Section 8.3.3.3 , Section 8.3.3.19.5 , Section 8.3.3.19.6

Table 8.12 AMS: Data Role Swap

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Data Role Swap, Initiated by UFP Operating as Sink (Accept)	1. DR_Swap Message 2. Accept Message	Started by Sink	Section 8.3.2.9.1.1	Section 8.3.3.19.1 , Section 8.3.3.19.2
Data Role Swap, Initiated by UFP Operating as Sink (Reject)	1. DR_Swap Message 2. Reject Message		Section 8.3.2.9.1.2	
Data Role Swap, Initiated by UFP Operating as Sink (Wait)	1. DR_Swap Message 2. Wait Message		Section 8.3.2.9.1.3	
Data Role Swap, Initiated by UFP Operating as Source (Accept)	1. DR_Swap Message 2. Accept Message	Started by Source	Section 8.3.2.9.2.1	
Data Role Swap, Initiated by UFP Operating as Source (Reject)	1. DR_Swap Message 2. Reject Message		Section 8.3.2.9.2.2	
Data Role Swap, Initiated by UFP Operating as Source (Wait)	1. DR_Swap Message 2. Wait Message		Section 8.3.2.9.2.3	
Data Role Swap, Initiated by DFP Operating as Source (Accept)	1. DR_Swap Message 2. Accept Message	Started by Source	Section 8.3.2.9.3.1	
Data Role Swap, Initiated by DFP Operating as Source (Reject)	1. DR_Swap Message 2. Reject Message		Section 8.3.2.9.3.2	
Data Role Swap, Initiated by DFP Operating as Source (Wait)	1. DR_Swap Message 2. Wait Message		Section 8.3.2.9.3.3	
Data Role Swap, Initiated by DFP Operating as Sink (Accept)	1. DR_Swap Message 2. Accept Message	Started by Sink	Section 8.3.2.9.4.1	
Data Role Swap, Initiated by DFP Operating as Sink (Reject)	1. DR_Swap Message 2. Reject Message		Section 8.3.2.9.4.2	
Data Role Swap, Initiated by DFP Operating as Sink (Wait)	1. DR_Swap Message 2. Wait Message		Section 8.3.2.9.4.3	

8.3.2.1.3.9

AMS: VCONN Swap

Table 8.13 AMS: VCONN Swap

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
VCONN Source Swap, initiated by VCONN Source (Accept)	1. VCONN_Swap Message 2. Accept Message 3. PS_RDY Message	Started by VCONN Source	Section 8.3.2.10.1.1	Section 8.3.3.20
VCONN Source Swap, initiated by VCONN Source (Reject)	1. VCONN_Swap Message 2. Reject Message		Section 8.3.2.10.1.2	
VCONN Source Swap, initiated by VCONN Source (Wait)	1. VCONN_Swap Message 2. Wait Message		Section 8.3.2.10.1.3	
VCONN Source Swap, initiated by non-VCONN Source (Accept)	1. VCONN_Swap Message 2. Accept Message 3. PS_RDY Message	Started by non-VCONN Source	Section 8.3.2.10.2.1	
VCONN Source Swap, initiated by non-VCONN Source (Reject)	1. VCONN_Swap Message 2. Reject Message		Section 8.3.2.10.2.2	
VCONN Source Swap, initiated by non-VCONN Source (Wait)	1. VCONN_Swap Message 2. Wait Message		Section 8.3.2.10.2.3	

8.3.2.1.3.10

AMS: Alert

Table 8.14 AMS: Alert

AMS	Message Sequence	Conditions	AMS Ref	AMS Ref
Source sends Alert to a Sink (SenderResponseTimer Timeout)	1. Alert Message	Started by Source	Section 8.3.2.11.1.1	Section 8.3.3.7.1, Section 8.3.3.7.2
Source sends Alert to a Sink (Get_Status Message)	1. Alert Message 2. Sink Gets Source Status AMS			
Sink sends Alert to a Source (SenderResponseTimer Timeout)	1. Alert Message	Started by Sink	Section 8.3.2.11.1.2	Section 8.3.3.7.3, Section 8.3.3.7.4
Sink sends Alert to a Source (Get_Status Message)	1. Alert Message 2. Source Gets Sink Status AMS			

Table 8.15 AMS: Status

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Sink Gets Source Status	1. <i>Get_Status</i> Message 2. <i>Status</i> Message	Started by Sink	Section 8.3.2.11.2.1 ,	Section 8.3.3.10.1 , Section 8.3.3.10.2
Source Gets Sink Status	1. <i>Get_Status</i> Message 2. <i>Status</i> Message	Started by Source	Section 8.3.2.11.2.2	
VCONN Source Gets Cable Plug Status	1. <i>Get_Status</i> Message 2. <i>Status</i> Message	Started by VCONN Source	Section 8.3.2.11.2.3 ,	
Sink Gets Source PPS Status	1. <i>Get_PPS_Status</i> Message 2. <i>PPS_Status</i> Message	Started by Sink	Section 8.3.2.11.2.4	Section 8.3.3.10.3 , Section 8.3.3.10.4

Table 8.16 AMS: Source/Sink Capabilities (SPR)

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Sink Gets Source Capabilities (EPR Mode)	1. <i>Get_Source_Cap</i> Message 2. <i>Source_Capabilities</i> Message	Started by Sink	Section 8.3.2.11.3.1.1 , Section 8.3.2.2.1.3.1 , Section 8.3.2.2.1.3.2 , Section 8.3.2.2.1.3.3	Section 8.3.3.2 , Section 8.3.3.3
Sink Gets Source Capabilities (Accept in SPR Mode)	1. <i>Get_Source_Cap</i> Message 2. <i>Source_Capabilities</i> Message 3. In SPR Mode only: SPR Sink Makes Request (Accept) AMS			
Sink Gets Source Capabilities (Reject in SPR Mode)	1. <i>Get_Source_Cap</i> Message 2. <i>Source_Capabilities</i> Message 3. In SPR Mode only: SPR Sink Makes Request (Reject) AMS			
Sink Gets Source Capabilities (Wait in SPR Mode)	1. <i>Get_Source_Cap</i> Message 2. <i>Source_Capabilities</i> Message 3. In SPR Mode only: SPR Sink Makes Request (Wait) AMS			
Dual-Role Power Source Gets Source Capabilities from a Dual-Role Power Sink	1. <i>Get_Source_Cap</i> Message 2. <i>Source_Capabilities</i> Message	Started by Source	Section 8.3.2.11.3.1.2	Section 8.3.3.19.7 , Section 8.3.3.19.10
Source Gets Sink Capabilities	1. <i>Get_Sink_Cap</i> Message 2. <i>Sink_Capabilities</i> Message	Started by Source	Section 8.3.2.11.3.1.3	Section 8.3.3.2 , Section 8.3.3.3
Dual-Role Power Sink Get Sink Capabilities from a Dual-Role Power Source	1. <i>Get_Sink_Cap</i> Message 2. <i>Sink_Capabilities</i> Message	Started by Sink	Section 8.3.2.11.3.1.4	Section 8.3.3.19.9 , Section 8.3.3.19.8

Table 8.17 AMS: Source/Sink Capabilities (EPR)

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Sink Gets EPR Source Capabilities (SPR Mode)	1. <i>EPR_Get_Source_Cap</i> Message 2. <i>EPR_Source_Capabilities</i> Message	Started by Sink	Section 8.3.2.11.3.2.1 , Section 8.3.2.2.2.5.1 , Section 8.3.2.2.2.5.2 , Section 8.3.2.2.2.5.3	Section 8.3.3.2 , Section 8.3.3.3 ,
Sink Gets EPR Source Capabilities (Accept in EPR Mode)	1. <i>EPR_Get_Source_Cap</i> Message 2. <i>EPR_Source_Capabilities</i> Message 3. In EPR Mode only: EPR Sink Makes Request (Accept) AMS			
Sink Gets EPR Source Capabilities (Reject in EPR Mode)	1. <i>EPR_Get_Source_Cap</i> Message 2. <i>EPR_Source_Capabilities</i> Message 3. In EPR Mode only: EPR Sink Makes Request (Reject) AMS			
Sink Gets EPR Source Capabilities (Wait in EPR Mode)	1. <i>EPR_Get_Source_Cap</i> Message 2. <i>EPR_Source_Capabilities</i> Message 3. In EPR Mode only: EPR Sink Makes Request (Wait) AMS			
Dual-Role Power Source Gets Source Capabilities from a Dual-Role Power EPR Sink	1. <i>EPR_Get_Source_Cap</i> Message 2. <i>EPR_Source_Capabilities</i> Message	Started by Source	Section 8.3.2.11.3.2.2	Section 8.3.3.19.7 , Section 8.3.3.19.10
Source Gets Sink EPR Capabilities	1. <i>EPR_Get_Sink_Cap</i> Message 2. <i>EPR_Sink_Capabilities</i> Message	Started by Source	Section 8.3.2.11.3.2.3	Section 8.3.3.2 , Section 8.3.3.3 ,
Dual-Role Power Sink Get Sink EPR Capabilities from a Dual-Role Power Source	1. <i>EPR_Get_Sink_Cap</i> Message 2. <i>EPR_Sink_Capabilities</i> Message	Started by Sink	Section 8.3.2.11.3.2.4	Section 8.3.3.19.8 , Section 8.3.3.19.9

8.3.2.1.3.14

AMS: Extended Capabilities

Table 8.18 AMS: Extended Capabilities

AMS	Interruptible	Message Sequence	Conditions	AMS Ref
Sink Gets Source Extended Capabilities	1. Get_Source_Cap_Extended Message 2. Source_Capabilities_Extended Message	Started by Sink	Section 8.3.2.11.4.1	Section 8.3.3.8.1 , Section 8.3.3.8.2
Dual-Role Power Source Gets Source Extended Capabilities from a Dual-Role Power Sink	1. Get_Source_Cap_Extended Message 2. Source_Capabilities_Extended Message	Started by Source	Section 8.3.2.11.4.2	Section 8.3.3.19.11 , Section 8.3.3.19.12
Source Gets Sink Extended Capabilities	1. Get_Sink_Cap_Extended Message 2. Sink_Capabilities_Extended Message	Started by Source	Section 8.3.2.11.4.3	Section 8.3.3.8.3 , Section 8.3.3.8.4
Dual-Role Power Sink Gets Sink Extended Capabilities from a Dual-Role Power Source	1. Get_Sink_Cap_Extended Message 2. Sink_Capabilities_Extended Message	Started by Sink	Section 8.3.2.11.4.4	Section 8.3.3.19.13 , Section 8.3.3.19.14

8.3.2.1.3.15

AMS: Battery Capabilities

Table 8.19 AMS: Battery Capabilities

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Sink Gets Battery Capabilities	1. Get_Battery_Cap Message 2. Battery_Capabilities Message	Started by Sink	Section 8.3.2.11.5.1	Section 8.3.3.11.1 , Section 8.3.3.11.2
Source Gets Battery Capabilities	1. Get_Battery_Cap Message 2. Battery_Capabilities Message	Started by Source	Section 8.3.2.11.5.2	
Sink Gets Battery Status	1. Get_Battery_Status Message 2. Battery_Status Message	Started by Sink	Section 8.3.2.11.5.3	Section 8.3.3.12.1 , Section 8.3.3.12.2
Sink Gets Battery Status	1. Get_Battery_Cap Message 2. Battery_Status Message	Started by Sink	Section 8.3.2.11.5.4	

8.3.2.1.3.16

AMS: Manufacturer Information

Table 8.20 AMS: Manufacturer Information

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source Gets Port Manufacturer Information from a Sink	1. Get_Manufacturer_Info Message 2. Manufacturer_Info Message	Started by Source	Section 8.3.2.11.6.1	Section 8.3.3.12.1 , Section 8.3.3.12.2
Sink Gets Port Manufacturer Information from a Source	1. Get_Manufacturer_Info Message 2. Manufacturer_Info Message	Started by Sink	Section 8.3.2.11.6.2	
Source Gets Battery Manufacturer Information from a Sink	1. Get_Manufacturer_Info Message 2. Manufacturer_Info Message	Started by Source	Section 8.3.2.11.6.3	
Sink Gets Battery Manufacturer Information from a Source	1. Get_Manufacturer_Info Message 2. Manufacturer_Info Message	Started by Sink	Section 8.3.2.11.6.4	
VCONN Source Gets Manufacturer Information from a Cable Plug	1. Get_Manufacturer_Info Message 2. Manufacturer_Info Message	Started by VCONN Source	Section 8.3.2.11.6.5	

8.3.2.1.3.17

AMS: Country Codes

Table 8.21 AMS: Country Codes

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source Gets Country Codes from a Sink	1. Get_Country_Codes Message 2. Country_Codes Message	Started by Source	Section 8.3.2.11.7.1	Section 8.3.3.14.1 , Section 8.3.3.14.2
Sink Gets Country Codes from a Source	1. Get_Country_Codes Message 2. Country_Codes Message	Started by Sink	Section 8.3.2.11.7.2	
VCONN Source Gets Country Codes from a Cable Plug	1. Get_Country_Codes Message 2. Country_Codes Message	Started by VCONN Source	Section 8.3.2.11.7.3	

8.3.2.1.3.18

AMS: Country Information

Table 8.22 AMS: Country Information

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source Gets Country Information from a Sink	1. Get_Country_Info Message 2. Country_Info Message	Started by Source	Section 8.3.2.11.8.1	Section 8.3.3.14.3 , Section 8.3.3.14.4
Sink Gets Country Information from a Source	1. Get_Country_Info Message 2. Country_Info Message	Started by Sink	Section 8.3.2.11.8.2	
VCONN Source Gets Country Information from a Cable Plug	1. Get_Country_Info Message 2. Country_Info Message	Started by VCONN Source	Section 8.3.2.11.8.3	

8.3.2.1.3.19

AMS: Revision Information

Table 8.23 AMS: Revision Information

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source Gets Revision Information from a Sink	1. <i>Get_Revision</i> Message 2. <i>Revision</i> Message	Started by Source	Section 8.3.2.11.9.1	Section 8.3.3.15.1 , Section 8.3.3.15.2
Sink Gets Revision Information from a Source	1. <i>Get_Revision</i> Message 2. <i>Revision</i> Message	Started by Sink	Section 8.3.2.11.9.2	
VCONN Source Gets Revision Information from a Cable Plug	1. <i>Get_Revision</i> Message 2. <i>Revision</i> Message	Started by VCONN Source	Section 8.3.2.11.9.1	

8.3.2.1.3.20

AMS: Source Information

Table 8.24 AMS: Source Information

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Sink Gets Source Information	1. <i>Get_Source_Cap_Extended</i> Message 2. <i>Source_Capabilities_Extended</i> Message	Started by Sink	Section 8.3.2.11.10.1	Section 8.3.3.9.1 , Section 8.3.3.9.2
Dual-Role Power Source Gets Source Information from a Dual-Role Power Sink	1. <i>Get_Source_Cap_Extended</i> Message 2. <i>Source_Capabilities_Extended</i> Message	Started by Source	Section 8.3.2.11.10.2	Section 8.3.3.19.15 , Section 8.3.3.19.16

8.3.2.1.3.21

AMS: Security

Table 8.25 AMS: Security

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source requests security exchange with Sink	1. <i>Security_Request</i> Message	Started by Source	Section 8.3.2.12.1	Section 8.3.3.17.1 , Section 8.3.3.17.2 , Section 8.3.3.17.3
Sink requests security exchange with Source	1. <i>Security_Request</i> Message	Started by Sink	Section 8.3.2.12.2	
VCONN Source requests security exchange with Cable Plug	1. <i>Security_Request</i> Message	Started by VCONN Source	Section 8.3.2.12.3	
Source responds to security exchange with Sink	1. <i>Security_Response</i> Message	Started by Source	Section 8.3.2.12.1	
Sink responds to security exchange with Source	1. <i>Security_Response</i> Message	Started by Sink	Section 8.3.2.12.2	
VCONN Source requests security exchange with Cable Plug	1. <i>Security_Response</i> Message	Started by VCONN Source	Section 8.3.2.12.3	

Table 8.26 AMS: Firmware Update

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source requests firmware update exchange with Sink	1. <i>Firmware_Update_Request Message</i>	Started by Source	Section 8.3.2.13.1	Section 8.3.3.18.1 , Section 8.3.3.18.2 , Section 8.3.3.18.3
Sink requests firmware update exchange with Source	1. <i>Firmware_Update_Request Message</i>	Started by Sink	Section 8.3.2.13.2	
VCONN Source requests firmware update exchange with Cable Plug	1. <i>Firmware_Update_Request Message</i>	Started by VCONN Source	Section 8.3.2.13.3	
Source responds to firmware update exchange with Sink	1. <i>Firmware_Update_Response Message</i>	Started by Source	Section 8.3.2.13.1	
Sink responds to firmware update exchange with Source	1. <i>Firmware_Update_Response Message</i>	Started by Sink	Section 8.3.2.13.2	
VCONN Source responds to firmware update exchange with Cable Plug	1. <i>Firmware_Update_Response Message</i>	Started by VCONN Source	Section 8.3.2.13.3	

Table 8.27 AMS: Structured VDM

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
Initiator to Responder Discover Identity (ACK)	1. Discover Identity REQ Command 2. Discover Identity ACK Command	Started by Initiator	Section 8.3.2.14.1.1	Section 8.3.3.21.1 , Section 8.3.3.22.1
Initiator to Responder Discover Identity (NAK)	1. Discover Identity REQ Command 2. Discover Identity NAK Command		Section 8.3.2.14.1.2	
Initiator to Responder Discover Identity (BUSY)	1. Discover Identity REQ Command 2. Discover Identity BUSY Command		Section 8.3.2.14.1.3	
Initiator to Responder Discover SVIDs (ACK)	1. Discover SVIDs REQ Command 2. Discover SVIDs ACK Command		Section 8.3.2.14.2.1	Section 8.3.3.21.2 , Section 8.3.3.22.2
Initiator to Responder Discover SVIDs (NAK)	1. Discover SVIDs REQ Command 2. Discover SVIDs NAK Command		Section 8.3.2.14.2.2	
Initiator to Responder Discover SVIDs (BUSY)	1. Discover SVIDs REQ Command 2. Discover SVIDs BUSY Command		Section 8.3.2.14.2.3	
Initiator to Responder Discover Modes (ACK)	1. Discover Modes REQ Command 2. Discover Modes ACK Command		Section 8.3.2.14.3.1	
Initiator to Responder Discover Modes (NAK)	1. Discover Modes REQ Command 2. Discover Modes NAK Command		Section 8.3.2.14.3.2	Section 8.3.3.21.3 , Section 8.3.3.22.3
Initiator to Responder Discover Modes (BUSY)	1. Discover Modes REQ Command 2. Discover Modes BUSY Command		Section 8.3.2.14.3.3	
DFP to UFP Enter Mode	1. Enter Mode REQ Command 2. Enter Mode ACK Command	Started by DFP	Section 8.3.2.14.4.1	Section 8.3.3.23.1 , Section 8.3.3.24.1
DFP to UFP Exit Mode	1. Exit Mode REQ Command 2. Exit Mode ACK Command		Section 8.3.2.14.4.2	Section 8.3.3.23.2 , Section 8.3.3.24.2
DFP to Cable Plug Enter Mode	1. Enter Mode REQ Command 2. Enter Mode ACK Command		Section 8.3.2.14.4.3	Section 8.3.3.23.1 , Section 8.3.3.25.4.1
DFP to Cable Plug Exit Mode	1. Exit Mode REQ Command 2. Exit Mode ACK Command		Section 8.3.2.14.4.4	Section 8.3.3.23.2 , Section 8.3.3.25.4.2
Initiator to Responder Attention	1. Attention REQ Command	Started by Initiator	Section 8.3.2.14.4.5	Section 8.3.3.21.4 , Section 8.3.3.22.4

8.3.2.1.3.24

AMS: Built-In Self-Test (BIST)

Table 8.28 AMS: Built-In Self-Test (BIST)

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
<i>BIST Carrier Mode</i>	1. BIST (BIST Carrier Mode) Message	Started by Tester	Section 8.3.2.15.1	Section 8.3.3.27.1
<i>BIST Test Data Mode</i>	1. BIST (BIST Test Data) Message		Section 8.3.2.15.2	Section 8.3.3.27.2
<i>BIST Shared Capacity Test Mode</i>	1. BIST (BIST Shared Test Mode Entry) Message 2. Series of Messages 3. BIST (BIST Shared Test Mode Exit) Message		Section 8.3.2.15.3	Section 8.3.3.27.3

8.3.2.1.3.25

AMS: Enter USB

Table 8.29 AMS: Enter USB

AMS	Message Sequence	Conditions	AMS Ref	State Machine Ref
<i>UFP Entering USB4[®] Mode (Accept)</i>	1. Enter_USB Message 2. Accept Message	Started by DFP	Section 8.3.2.16.1.1	Section 8.3.3.16.1 , Section 8.3.3.16.2
<i>UFP Entering USB4 Mode (Reject)</i>	1. Enter_USB Message 2. Reject Message		Section 8.3.2.16.1.2	
<i>UFP Entering USB4 Mode (Wait)</i>	1. Enter_USB Message 2. Wait Message		Section 8.3.2.16.1.3	
<i>Cable Plug Entering USB4 Mode (Accept)</i>	1. Enter_USB Message 2. Accept Message		Section 8.3.2.16.2.1	
<i>Cable Plug Entering USB4 Mode (Reject)</i>	1. Enter_USB Message 2. Reject Message		Section 8.3.2.16.2.2	
<i>Cable Plug Entering USB4 Mode (Wait)</i>	1. Enter_USB Message 2. Wait Message		Section 8.3.2.16.2.3	

8.3.2.1.3.26

AMS: Unstructured VDM

Table 8.30 AMS: Unstructured VDM

AMS	Message Sequence	AMS Ref	State Machine Ref
<i>Unstructured VDM</i>	1. Unstructured Vendor_Defined Message	Section 8.3.2.17.1	
<i>VDEM</i>	1. Vendor_Defined_Extended Message	Section 8.3.2.17.2	

Table 8.31 AMS: Hard Reset

AMS	Interruptible	Message Sequence	Conditions	AMS Ref	State Machine Ref
Source Initiated <i>Hard Reset</i>	No	1. Hard Reset Signaling 2. SPR Explicit Contract Negotiation AMS	Started by <i>Source</i>	Section 8.3.2.6.1 , Section 8.3.2.2.1.1	Section 8.3.3.2 , Section 8.3.3.3
Sink Initiated <i>Hard Reset</i>	No	1. Hard Reset Signaling 2. SPR Explicit Contract Negotiation AMS	Started by <i>Sink</i>	Section 8.3.2.6.2 , Section 8.3.2.2.1.1	
Source Initiated <i>Hard Reset – Sink Long Reset</i>	No	1. Hard Reset Signaling 2. SPR Explicit Contract Negotiation AMS	Started by <i>Source</i>	Section 8.3.2.6.3 , Section 8.3.2.2.1.1	

8.3.2.2 Power Negotiation

8.3.2.2.1 SPR

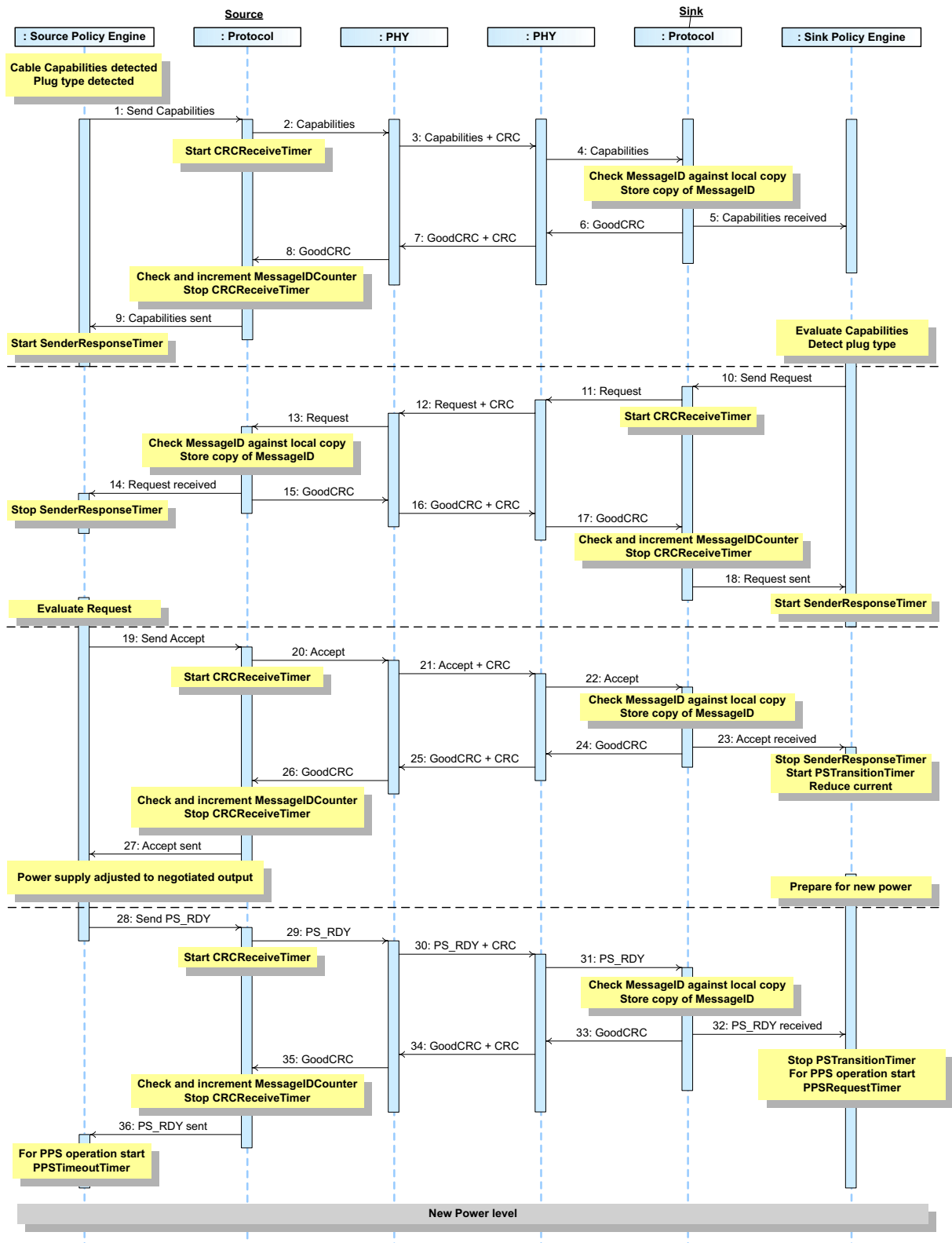
8.3.2.2.1.1 SPR Explicit Contract Negotiation

8.3.2.2.1.1.1 SPR Explicit Contract Negotiation (Accept)

Figure 8.5, "Successful Fixed, Variable or Battery SPR Power Negotiation" illustrates an example of a successful Message flow while negotiating an *Explicit Contract* in *SPR Mode*. The *Negotiation* goes through 5 distinct phases:

- The *Source* sends out its power *Capabilities* in a *Source_Capabilities* Message.
- The *Sink* evaluates these *Capabilities*, and, in the request, phase selects one power level by sending a *Request* Message.
- The *Source* evaluates the request and accepts the request with an *Accept* Message.
- The *Source* transitions to the new power level and then informs the *Sink* by sending a *PS_RDY* Message.
- The *Sink* starts using the new power level.
- For *SPR PPS* operation:
 - the *Source* starts its keep alive timer.
 - the *Sink* starts its request timer to send periodic *Request* Messages.

Figure 8.5 Successful Fixed, Variable or Battery SPR Power Negotiation



[Table 8.32, "Steps for a successful Power Negotiation"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.5, "Successful Fixed, Variable or Battery SPR Power Negotiation"](#) above.

Table 8.32 Steps for a successful Power Negotiation

Step	Source	Sink
1	The <i>Cable Capabilities</i> or Plug Type are detected if these are not already known (see Section 4.4, "Cable Type Detection"). <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>Source_Capabilities</i> Message that represents the power supply's present capabilities.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the <i>Source_Capabilities</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Source_Capabilities</i> Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the <i>Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> evaluates the <i>Source_Capabilities</i> Message sent by the <i>Source</i> , detects the plug type if this is necessary (see Section 4.4, "Cable Type Detection") and selects which power it would like. It tells the <i>Protocol Layer</i> to form the data (e.g., <i>Power Data Object</i>) that represents its <i>Request</i> into a Message.
11		<i>Protocol Layer</i> creates the <i>Request</i> Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Request</i> Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the <i>Request</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>PHY Layer</i> removes the CRC and forwards the <i>Request</i> Message to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops <i>SenderResponseTimer</i> .	

Table 8.32 Steps for a successful Power Negotiation

Step	Source	Sink
15	The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends CRC and sends the Message.	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . It informs the <i>Policy Engine</i> that the Request Message was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer . The <i>Policy Engine</i> starts SenderResponseTimer .
19	<i>Policy Engine</i> evaluates the Request Message sent by the Sink and decides if it can meet the request. It tells the <i>Protocol Layer</i> to form an Accept Message.	
20	The <i>Protocol Layer</i> forms the Accept Message that is passed to the <i>PHY Layer</i> .	
21	<i>PHY Layer</i> appends CRC and sends the Accept Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.
22		<i>PHY Layer</i> forwards the Accept Message to the <i>Protocol Layer</i> .
23		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept Message has been received. The <i>Policy Engine</i> stops SenderResponseTimer , starts the PSTransitionTimer and reduces its current draw. The <i>DPM</i> prepares the Power supply for transition to the new power level.
24		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
25	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends CRC and sends the Message.
26	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
27	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept Message was successfully sent.	
Power supply Adjusts its Output to the <i>Negotiated</i> Value		
28	The <i>DPM</i> informs the <i>Policy Engine</i> that the power supply has settled at the new operating condition and tells the <i>Protocol Layer</i> to send a PS_RDY Message.	
29	The <i>Protocol Layer</i> forms the PS_RDY Message.	
30	<i>PHY Layer</i> appends CRC and sends the PS_RDY Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PS_RDY Message and compares the CRC it calculated with the one sent to verify the Message.
31		<i>PHY Layer</i> forwards the PS_RDY Message to the <i>Protocol Layer</i> .

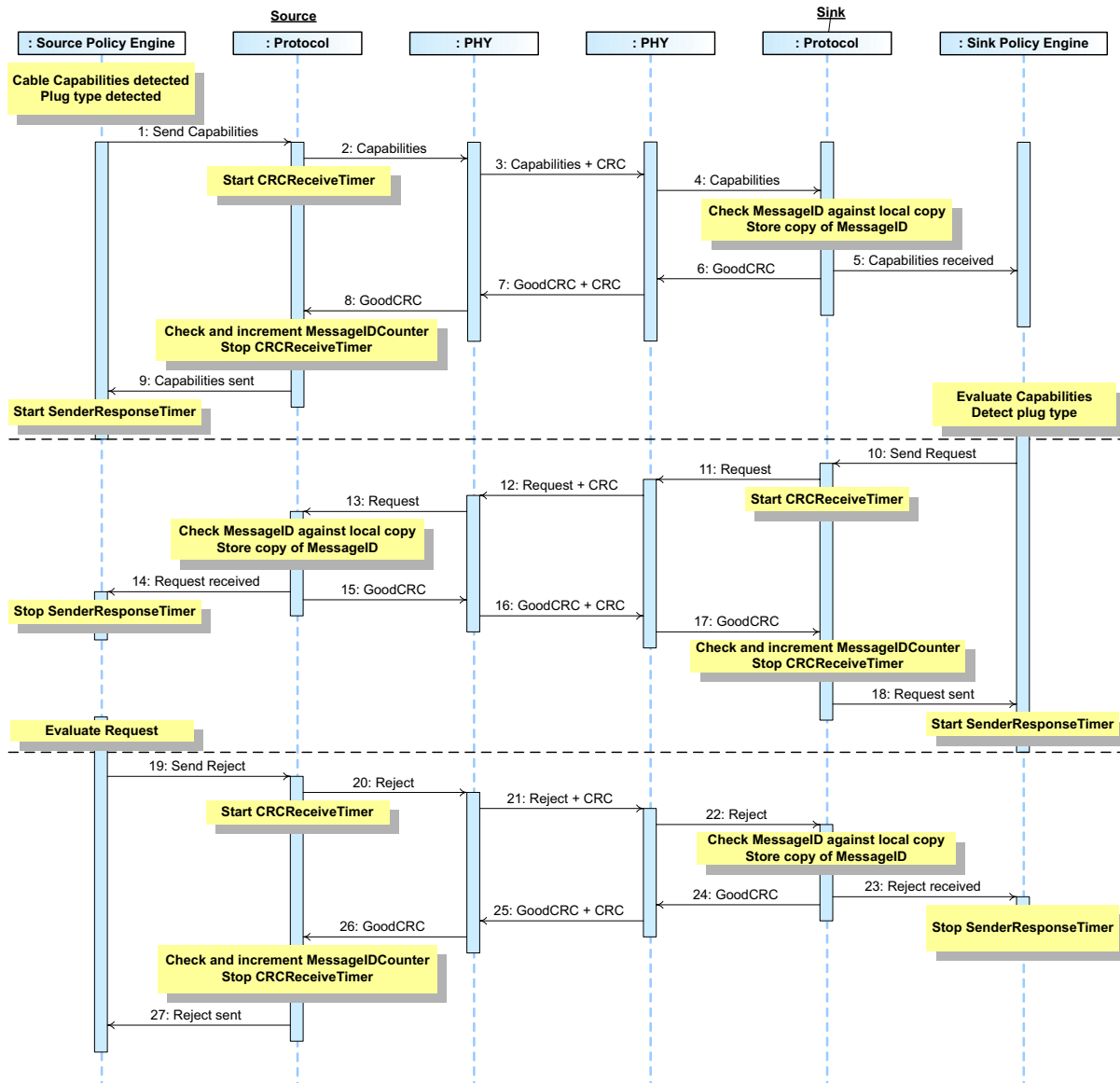
Table 8.32 Steps for a successful Power Negotiation

Step	Source	Sink
32		<p><i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that a RS_RDY has been received. The <i>Policy Engine</i> stops the PSTransitionTimer.</p> <p>When in <i>SPR PPS Mode</i> the <i>Policy Engine</i> starts the SinkPPSPeriodicTimer.</p>
33		The <i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it to its <i>PHY Layer</i> .
34	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> .
35	<i>PHY Layer</i> forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . Stops the CRCReceiveTimer .	
36	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY <i>Message</i> was successfully sent.	
37	When in <i>SPR PPS Mode</i> the <i>Policy Engine</i> starts the SourcePPSCommTimer .	
New Power Level Negotiated		

Figure 8.6, "Rejected Fixed, Variable or Battery SPR Power Negotiation" illustrates an example of a Message flow where the request is rejected while negotiating an *Explicit Contract* in *SPR Mode*. The *Negotiation* goes through the following phases:

- The *Source* sends out its power *Capabilities* in a *Source_Capabilities* Message.
- The *Sink* evaluates these *Capabilities*, and, in the request, phase selects one power level by sending a *Request* Message.
- The *Source* evaluates the request and rejects the request with a *Reject* Message.

Figure 8.6 Rejected Fixed, Variable or Battery SPR Power Negotiation



[Table 8.33, "Steps for a rejected Power Negotiation"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.6, "Rejected Fixed, Variable or Battery SPR Power Negotiation"](#) above.

Table 8.33 Steps for a rejected Power Negotiation

Step	Source	Sink
1	The <i>Cable Capabilities</i> or Plug Type are detected if these are not already known (see Section 4.4, "Cable Type Detection"). <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>Source_Capabilities</i> Message that represents the power supply's present capabilities.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Source_Capabilities</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Source_Capabilities</i> Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> evaluates the <i>Source_Capabilities</i> Message sent by the <i>Source</i> , detects the plug type if this is necessary (see Section 4.4, "Cable Type Detection") and selects which power it would like. It tells the <i>Protocol Layer</i> to form the data (e.g., <i>Power Data Object</i>) that represents its Request into a Message.
11		<i>Protocol Layer</i> creates the <i>Request</i> Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Request</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Request</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Request</i> Message to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops <i>SenderResponseTimer</i> .	

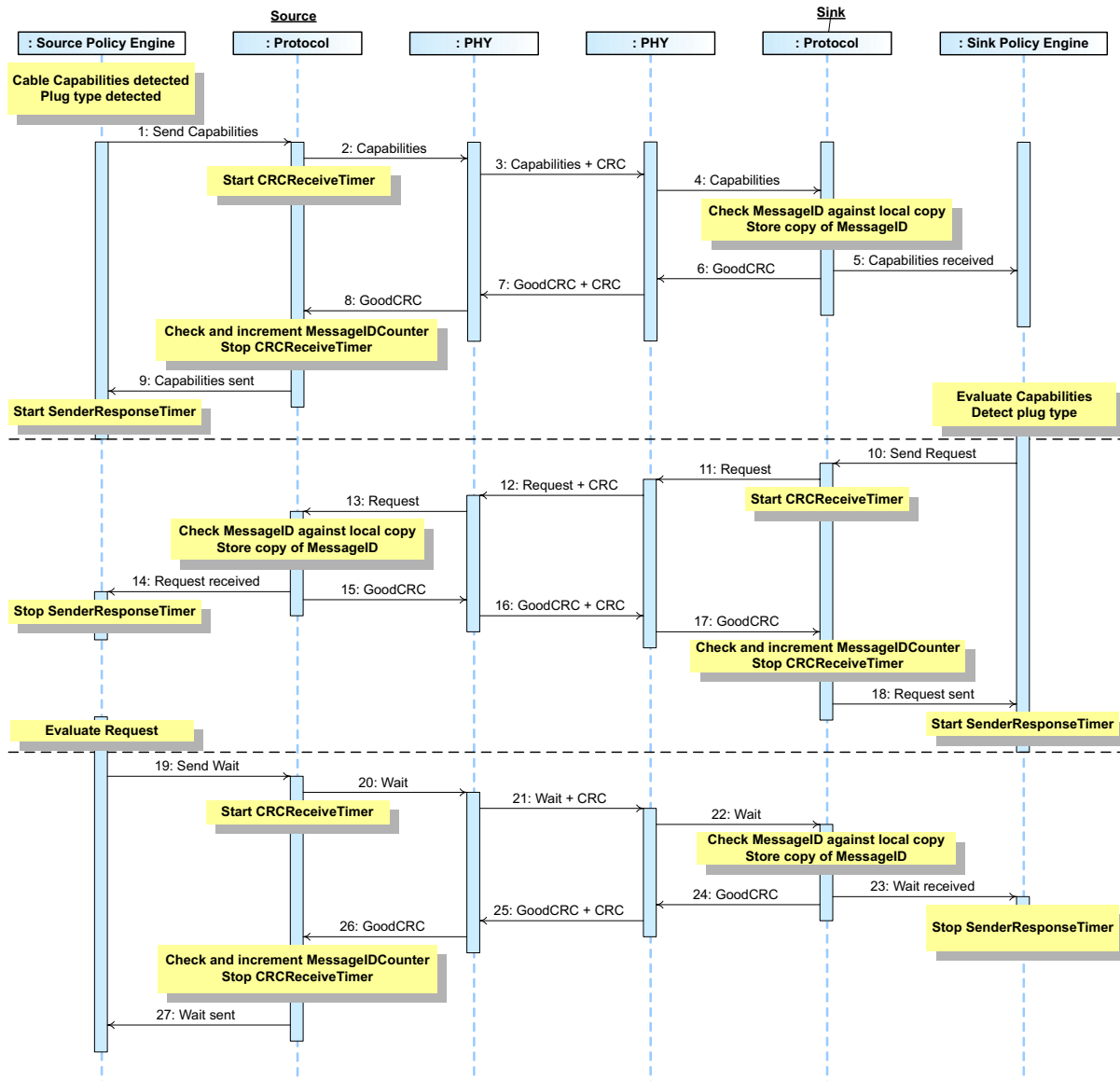
Table 8.33 Steps for a rejected Power Negotiation

Step	Source	Sink
15	The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends CRC and sends the Message.	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . It informs the <i>Policy Engine</i> that the Request Message was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer . The <i>Policy Engine</i> starts SenderResponseTimer .
19	<i>Policy Engine</i> evaluates the Request Message sent by the <i>Sink</i> and decides it can't meet the request. It tells the <i>Protocol Layer</i> to form a Reject Message.	
20	The <i>Protocol Layer</i> forms the Reject Message that is passed to the <i>PHY Layer</i> .	
21	<i>PHY Layer</i> appends CRC and sends the Reject Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.
22		<i>PHY Layer</i> forwards the Reject Message to the <i>Protocol Layer</i> .
23		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Reject Message has been received. The <i>Policy Engine</i> stops SenderResponseTimer .
24		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
25	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends CRC and sends the Message.
26	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
27	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Reject Message was successfully sent.	

Figure 8.7, "Wait response to Fixed, Variable or Battery SPR Power Negotiation" illustrates an example of a Message flow where the request is responded to with wait while negotiating an *Explicit Contract* in *SPR Mode*. The *Negotiation* goes through the following phases:

- The *Source* sends out its power *Capabilities* in a *Source_Capabilities* Message.
- The *Sink* evaluates these *Capabilities*, and, in the request, phase selects one power level by sending a *Request* Message.
- The *Source* evaluates the request and rejects the request with a *Wait* Message.

Figure 8.7 Wait response to Fixed, Variable or Battery SPR Power Negotiation



[Table 8.34, "Steps for a Wait response to a Power Negotiation"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.7, "Wait response to Fixed, Variable or Battery SPR Power Negotiation"](#) above.

Table 8.34 Steps for a Wait response to a Power Negotiation

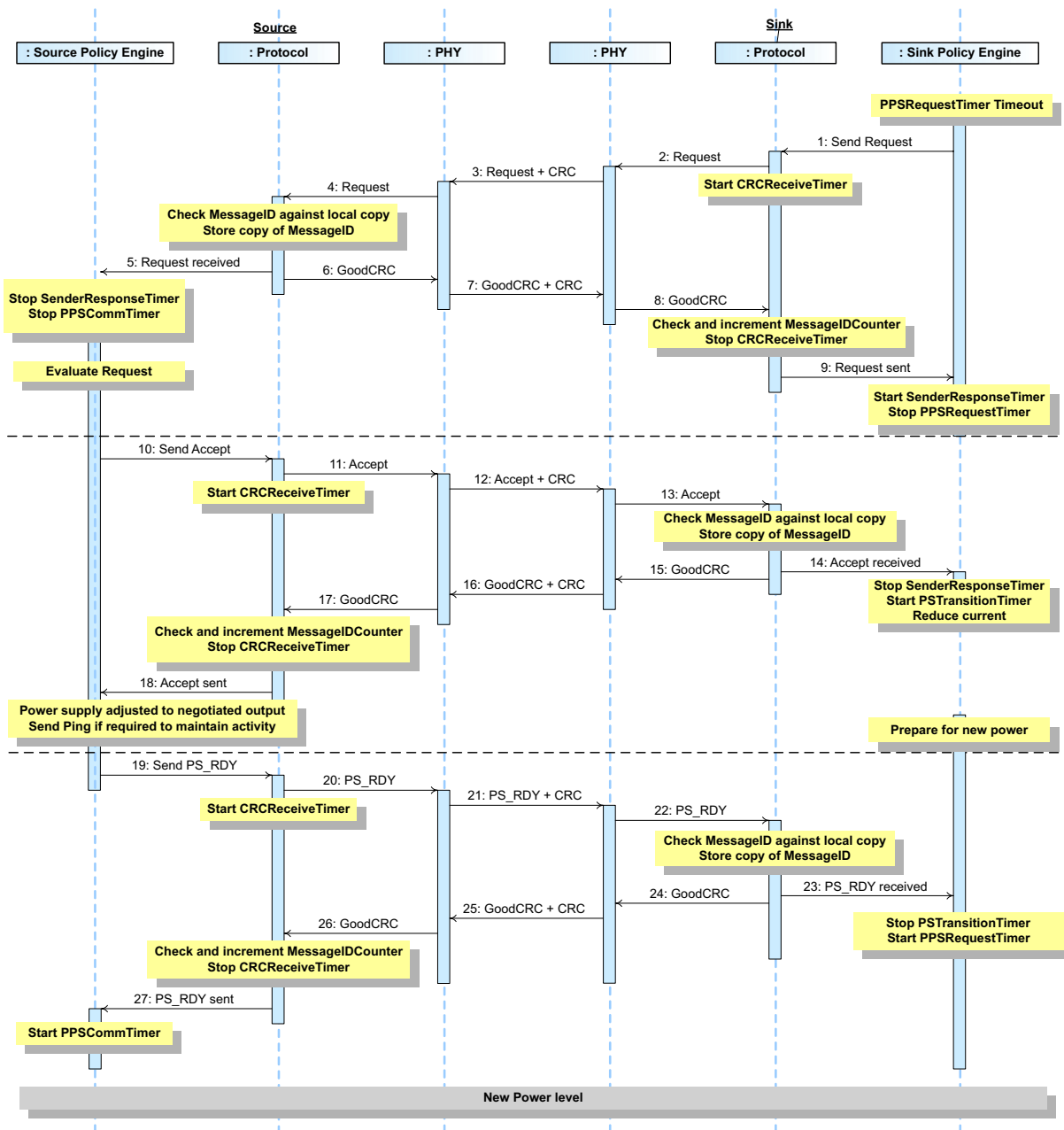
Step	Source	Sink
1	The <i>Cable Capabilities</i> or Plug Type are detected if these are not already known (see Section 4.4, "Cable Type Detection"). <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>Source_Capabilities</i> Message that represents the power supply's present capabilities.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the <i>Source_Capabilities</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Source_Capabilities</i> Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the <i>Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> evaluates the <i>Source_Capabilities</i> Message sent by the <i>Source</i> , detects the plug type if this is necessary (see Section 4.4, "Cable Type Detection") and selects which power it would like. It tells the <i>Protocol Layer</i> to form the data (e.g., <i>Power Data Object</i>) that represents its Request into a Message.
11		<i>Protocol Layer</i> creates the <i>Request</i> Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Request</i> Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the <i>Request</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>PHY Layer</i> removes the CRC and forwards the <i>Request</i> Message to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops <i>SenderResponseTimer</i> .	

Table 8.34 Steps for a Wait response to a Power Negotiation

Step	Source	Sink
15	The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends CRC and sends the Message.	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . It informs the <i>Policy Engine</i> that the Request Message was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer . The <i>Policy Engine</i> starts SenderResponseTimer .
19	<i>Policy Engine</i> evaluates the Request Message sent by the Sink and decides if it can meet the request. It tells the <i>Protocol Layer</i> to form a Wait Message.	
20	The <i>Protocol Layer</i> forms the Wait Message that is passed to the <i>PHY Layer</i> .	
21	<i>PHY Layer</i> appends CRC and sends the Wait Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.
22		<i>PHY Layer</i> forwards the Wait Message to the <i>Protocol Layer</i> .
23		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Wait Message has been received. The <i>Policy Engine</i> stops SenderResponseTimer .
24		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
25	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends CRC and sends the Message.
26	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
27	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Wait Message was successfully sent.	

This is an example of *SPR PPS* keep alive operation during an *Explicit Contract* with *SPR PPS* as the *APDO*. [Figure 8.8, "SPR PPS Keep Alive"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.8 SPR PPS Keep Alive



[Table 8.35. "Steps for SPR PPS Keep Alive"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.8. "SPR PPS Keep Alive"](#) above.

Table 8.35 Steps for SPR PPS Keep Alive

Step	Source	Sink
1		<p>The SinkPPSPeriodicTimer times out in the <i>Policy Engine</i>. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Request Message.</p> <p>The <i>Protocol Layer</i> creates the Request Message and passes it to <i>PHY Layer</i>.</p>
2	<i>PHY Layer</i> receives the Request Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Request Message. Starts CRCReceiveTimer .
3	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Request Message to the <i>Protocol Layer</i> .	
4	<p><i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i>. <i>Policy Engine</i> stops the SourcePPSCommTimer.</p>	
5	The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .	
6	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
7		<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> .
8		<p>The <i>Protocol Layer</i> verifies and increments the MessageIDCounter. It informs the <i>Policy Engine</i> that the Request Message was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer.</p> <p>The <i>Policy Engine</i> starts SenderResponseTimer.</p>
9	<i>Policy Engine</i> requests the <i>DPM</i> to evaluate the Request Message sent by the <i>Sink</i> and decides if the <i>Source</i> can meet the request. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.	
10	The <i>Protocol Layer</i> forms the Accept Message that is passed to the <i>PHY Layer</i> .	
11	<i>PHY Layer</i> appends <i>CRC</i> and sends the Accept Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Accept Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
12		<i>PHY Layer</i> forwards the Accept Message to the <i>Protocol Layer</i> .
13		<p><i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept Message has been received. The <i>Policy Engine</i> stops SenderResponseTimer, starts the PSTransitionTimer and reduces its current draw.</p> <p>The <i>DPM</i> prepares the Power supply for transition to the new power level.</p>

Table 8.35 Steps for SPR PPS Keep Alive

Step	Source	Sink
14		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept Message was successfully sent.	
Power supply Adjusts its Output to the <i>Negotiated</i> Value		
18	The <i>DPM</i> informs the <i>Policy Engine</i> that the power supply has settled at the new operating condition and tells the <i>Protocol Layer</i> to send a PS_RDY Message.	
19	The <i>Protocol Layer</i> forms the PS_RDY Message.	
20	<i>PHY Layer</i> appends CRC and sends the PS_RDY Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PS_RDY Message and compares the CRC it calculated with the one sent to verify the Message.
21		<i>PHY Layer</i> forwards the PS_RDY Message to the <i>Protocol Layer</i> .
22		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a RS_RDY has been received. The <i>Policy Engine</i> stops the PSTransitionTimer . When in <i>SPR PPS Mode</i> the <i>Policy Engine</i> starts the SinkPPSPeriodicTimer .
23		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
24	<i>PHY Layer</i> receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
25	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . Stops the CRCReceiveTimer .	
26	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent.	
27	When in <i>SPR PPS Mode</i> the <i>Policy Engine</i> starts the SourcePPSCommTimer .	

8.3.2.2.1.3

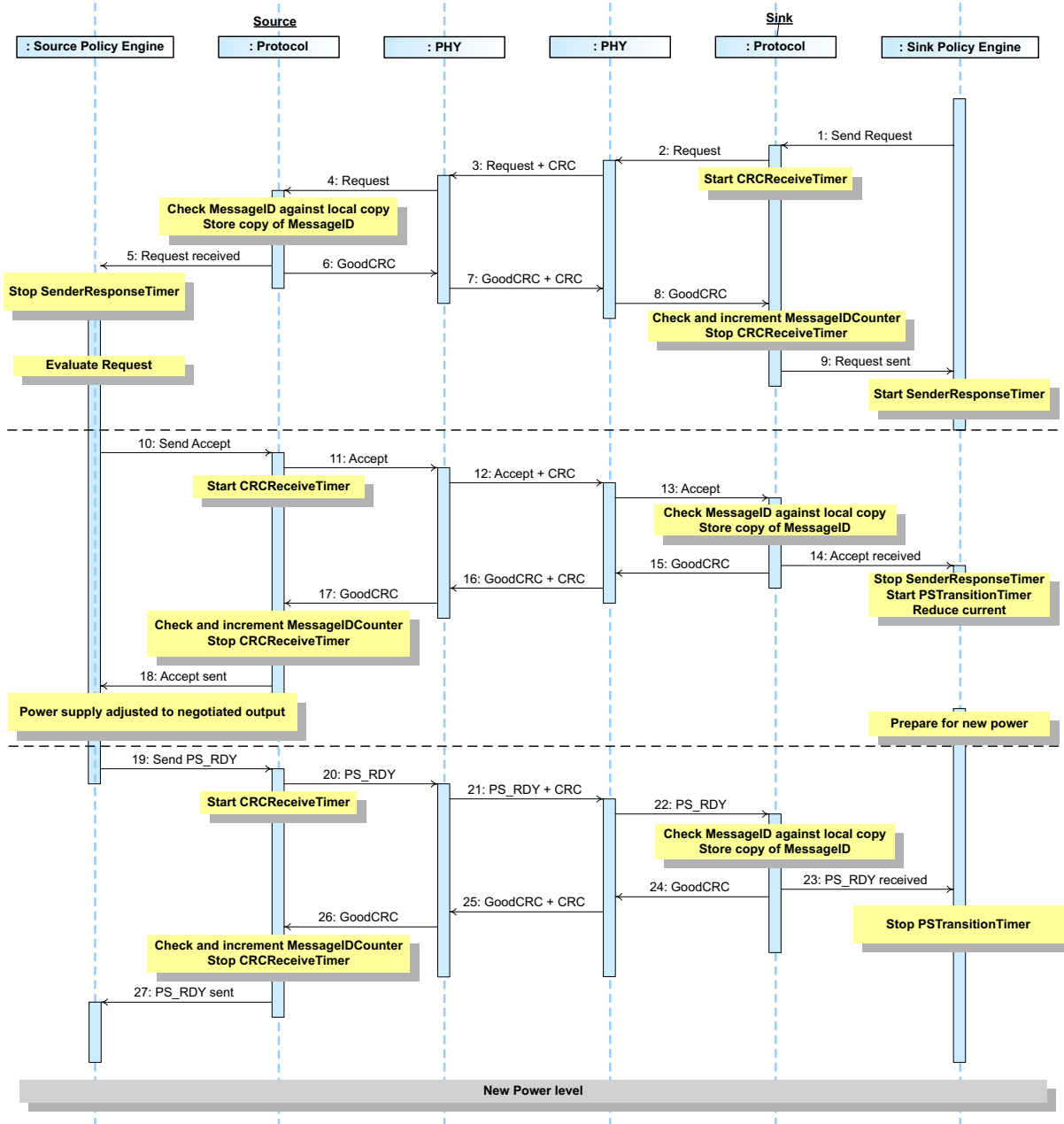
SPR Sink Makes Request

8.3.2.2.1.3.1

SPR Sink Makes Request (Accept)

This is an example of SPR when a *Sink* makes a Request which is Accepted during an *Explicit Contract*. [Figure 8.9, "SPR Sink Makes Request \(Accept\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.9 SPR Sink Makes Request (Accept)



[Table 8.36, "Steps for SPR Sink Makes Request \(Accept\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.9, "SPR Sink Makes Request \(Accept\)"](#) above.

Table 8.36 Steps for SPR Sink Makes Request (Accept)

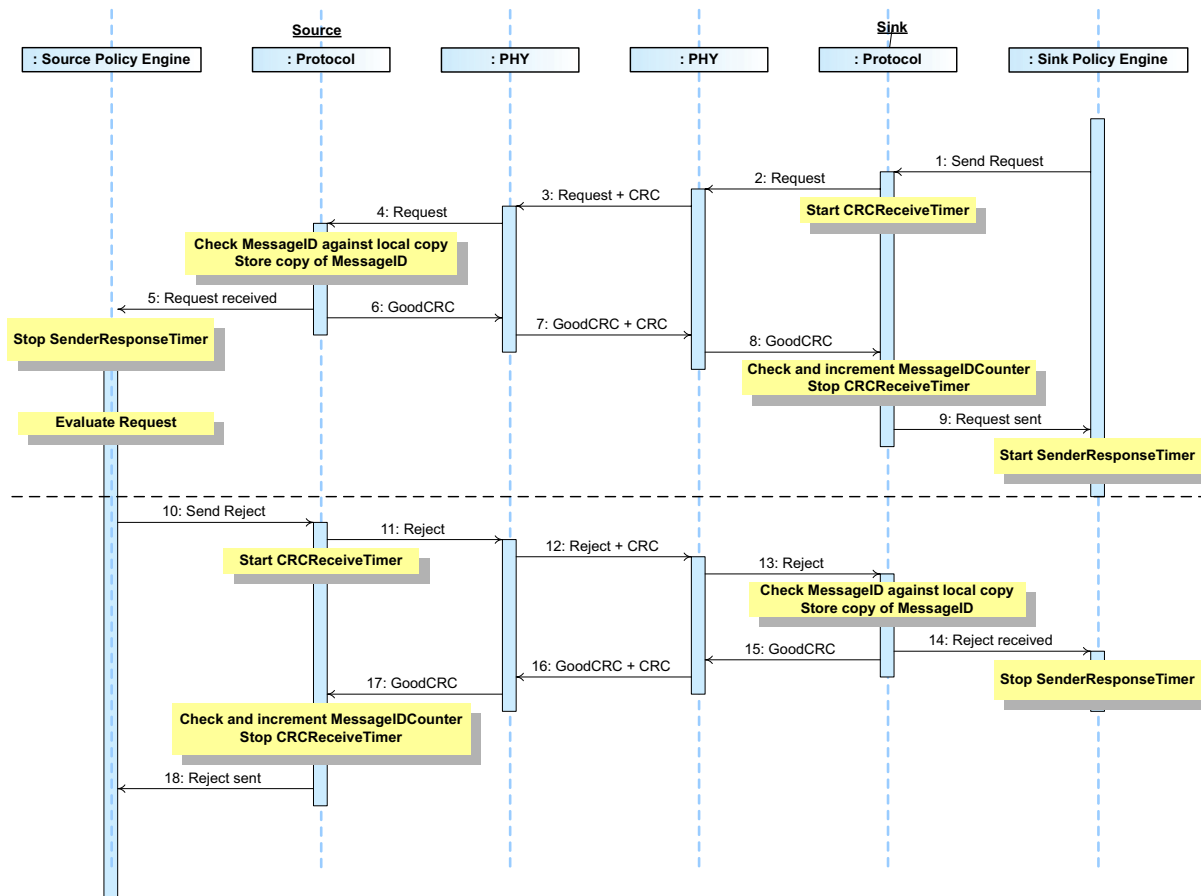
Step	Source	Sink
1		<i>DPM tells the Policy Engine to request a different power level. The Policy Engine tells the Protocol Layer to form a Request Message.</i> <i>The Protocol Layer creates the Request Message and passes it to PHY Layer.</i>
2	<i>PHY Layer receives the Request Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the Request Message. Starts CRCReceiveTimer.</i>
3	<i>PHY Layer removes the CRC and forwards the Request Message to the Protocol Layer.</i>	
4	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>The Protocol Layer passes the Request information to the Policy Engine.</i>	
5	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
6	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
7		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
8		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i> <i>The Policy Engine starts SenderResponseTimer.</i>
9	<i>Policy Engine requests the DPM to evaluate the Request Message sent by the Sink and decides if the Source can meet the request. The Policy Engine tells the Protocol Layer to form an Accept Message.</i>	
10	<i>The Protocol Layer forms the Accept Message that is passed to the PHY Layer.</i>	
11	<i>PHY Layer appends CRC and sends the Accept Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Accept Message and compares the CRC it calculated with the one sent to verify the Message.</i>
12		<i>PHY Layer forwards the Accept Message to the Protocol Layer.</i>
13		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>Protocol Layer informs the Policy Engine that an Accept Message has been received. The Policy Engine stops SenderResponseTimer, starts the PSTransitionTimer and reduces its current draw.</i> <i>The DPM prepares the Power supply for transition to the new power level.</i>

Table 8.36 Steps for SPR Sink Makes Request (Accept)

Step	Source	Sink
14		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept Message was successfully sent.	
Power supply Adjusts its Output to the <i>Negotiated</i> Value		
18	The <i>DPM</i> informs the <i>Policy Engine</i> that the power supply has settled at the new operating condition and tells the <i>Protocol Layer</i> to send a PS_RDY Message.	
19	The <i>Protocol Layer</i> forms the PS_RDY Message.	
20	<i>PHY Layer</i> appends <i>CRC</i> and sends the PS_RDY Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PS_RDY Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
21		<i>PHY Layer</i> forwards the PS_RDY Message to the <i>Protocol Layer</i> .
22		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a PS_RDY has been received. The <i>Policy Engine</i> stops the PSTransitionTimer .
23		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
24	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
25	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . Stops the CRCReceiveTimer .	
26	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent.	
New Power Level <i>Negotiated</i>		

This is an example of SPR when a *Sink* makes a Request which is Rejected during an *Explicit Contract*. [Figure 8.10, "SPR Sink Makes Request \(Reject\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.10 SPR Sink Makes Request (Reject)



[Table 8.37, "Steps for SPR Sink Makes Request \(Reject\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.10, "SPR Sink Makes Request \(Reject\)"](#) above.

Table 8.37 Steps for SPR Sink Makes Request (Reject)

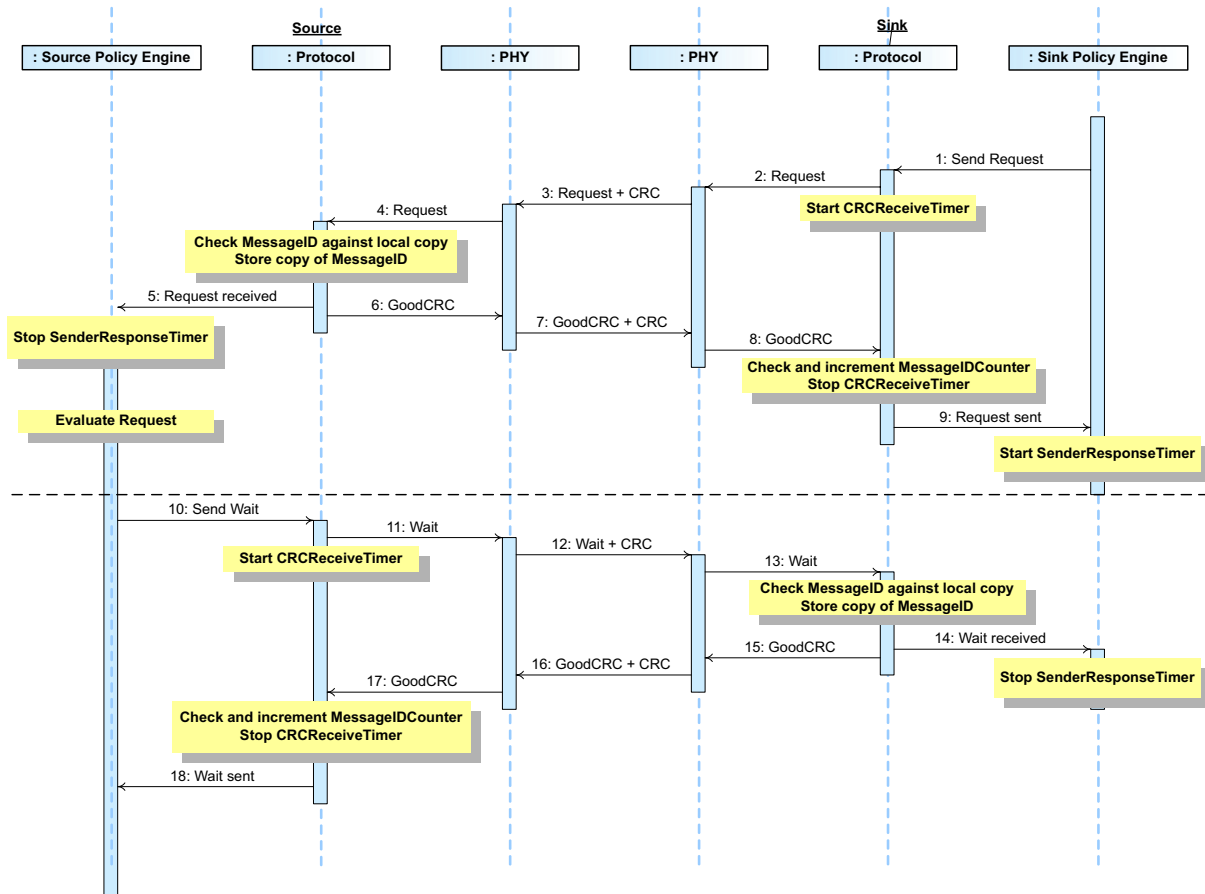
Step	Source	Sink
1		<i>DPM tells the Policy Engine to request a different power level. The Policy Engine tells the Protocol Layer to form a Request Message.</i> <i>The Protocol Layer creates the Request Message and passes it to PHY Layer.</i>
2	<i>PHY Layer receives the Request Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the Request Message. Starts CRCReceiveTimer.</i>
3	<i>PHY Layer removes the CRC and forwards the Request Message to the Protocol Layer.</i>	
4	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>The Protocol Layer passes the Request information to the Policy Engine.</i>	
5	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
6	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
7		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
8		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i> <i>The Policy Engine starts SenderResponseTimer.</i>
9	<i>Policy Engine requests the DPM to evaluate the Request Message sent by the Sink and decides that the Source can't meet the request. The Policy Engine tells the Protocol Layer to form a Reject Message.</i>	
10	<i>The Protocol Layer forms the Reject Message that is passed to the PHY Layer.</i>	
11	<i>PHY Layer appends CRC and sends the Reject Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Reject Message and compares the CRC it calculated with the one sent to verify the Message.</i>
12		<i>PHY Layer forwards the Reject Message to the Protocol Layer.</i>
13		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>Protocol Layer informs the Policy Engine that an Reject Message has been received. The Policy Engine informs the DPM that the Request has been rejected.</i>
14		<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>

Table 8.37 Steps for SPR Sink Makes Request (Reject)

Step	Source	Sink
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Reject Message was successfully sent.	

This is an example of SPR when a *Sink* makes a Request which is responded to with a *Wait Message* during an *Explicit Contract*. [Figure 8.11, "SPR Sink Makes Request \(Wait\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.11 SPR Sink Makes Request (Wait)



[Table 8.38, "Steps for SPR Sink Makes Request \(Wait\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.11, "SPR Sink Makes Request \(Wait\)"](#) above.

Table 8.38 Steps for SPR Sink Makes Request (Wait)

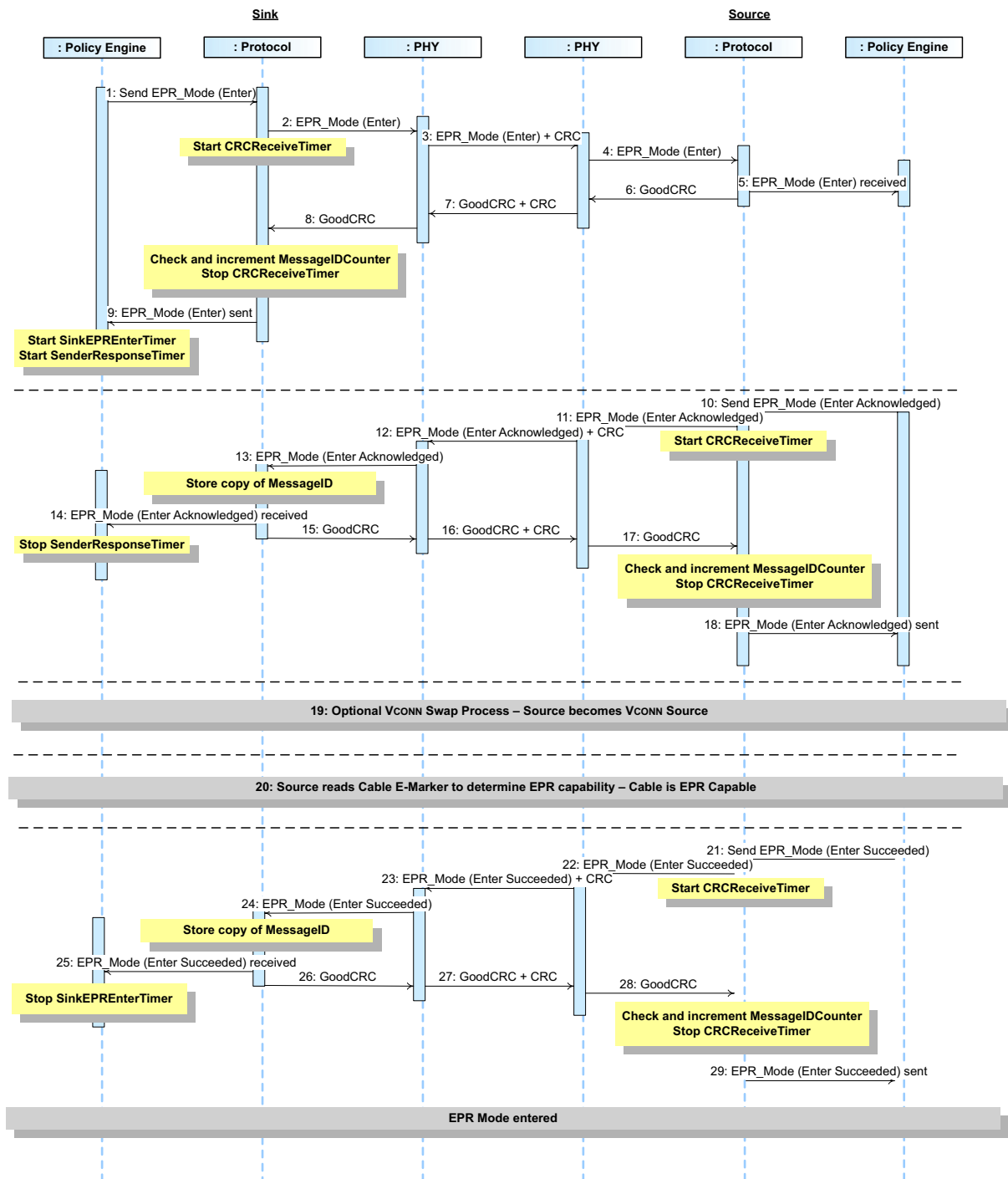
Step	Source	Sink
1		<i>DPM tells the Policy Engine to request a different power level. The Policy Engine tells the Protocol Layer to form a Request Message.</i> <i>The Protocol Layer creates the Request Message and passes it to PHY Layer.</i>
2	<i>PHY Layer receives the Request Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the Request Message. Starts CRCReceiveTimer.</i>
3	<i>PHY Layer removes the CRC and forwards the Request Message to the Protocol Layer.</i>	
4	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>The Protocol Layer passes the Request information to the Policy Engine.</i>	
5	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
6	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
7		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
8		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i> <i>The Policy Engine starts SenderResponseTimer.</i>
9	<i>Policy Engine requests the DPM to evaluate the Request Message sent by the Sink and decides if the Source can meet the request. The Policy Engine tells the Protocol Layer to form a Wait Message.</i>	
10	<i>The Protocol Layer forms the Wait Message that is passed to the PHY Layer.</i>	
11	<i>PHY Layer appends CRC and sends the Wait Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Wait Message and compares the CRC it calculated with the one sent to verify the Message.</i>
12		<i>PHY Layer forwards the Wait Message to the Protocol Layer.</i>
13		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>Protocol Layer informs the Policy Engine that an Wait Message has been received. The Policy Engine informs the DPM that the Request has been rejected.</i>
14		<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>

Table 8.38 Steps for SPR Sink Makes Request (Wait)

Step	Source	Sink
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Wait Message was successfully sent.	

8.3.2.2.2.1.1 Entering EPR Mode (Success)

Figure 8.12 Entering EPR Mode (Success)



[Table 8.39, "Steps for Entering EPR Mode \(Success\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.12, "Entering EPR Mode \(Success\)"](#) above.

Table 8.39 Steps for Entering EPR Mode (Success)

Step	Sink	Source
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an EPR_Mode (Enter) <i>Message</i> to request entry to <i>EPR Mode</i> .	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the EPR_Mode (Enter) <i>Message</i> . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Mode (Enter) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Mode (Enter) <i>Message</i> to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Mode (Enter) <i>Message</i> information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Mode (Enter) Source_Capabilities <i>Message</i> was successfully sent. The <i>Policy Engine</i> starts the SenderResponseTimer and the SinkEPREnterTimer .	
10		<i>Policy Engine</i> evaluates the EPR_Mode (Enter) <i>Message</i> sent by the <i>Sink</i> . It tells the <i>Protocol Layer</i> to form a EPR_Mode (Enter Acknowledged) <i>Message</i> .
11		<i>Protocol Layer</i> creates the EPR_Mode (Enter Acknowledged) <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the EPR_Mode (Enter Acknowledged) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the EPR_Mode (Enter Acknowledged) <i>Message</i> . Starts CRCReceiveTimer .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Mode (Enter Acknowledged) <i>Message</i> to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the EPR_Mode (Enter Acknowledged) information to the <i>Policy Engine</i> . The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	The <i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it to its <i>PHY Layer</i> .	

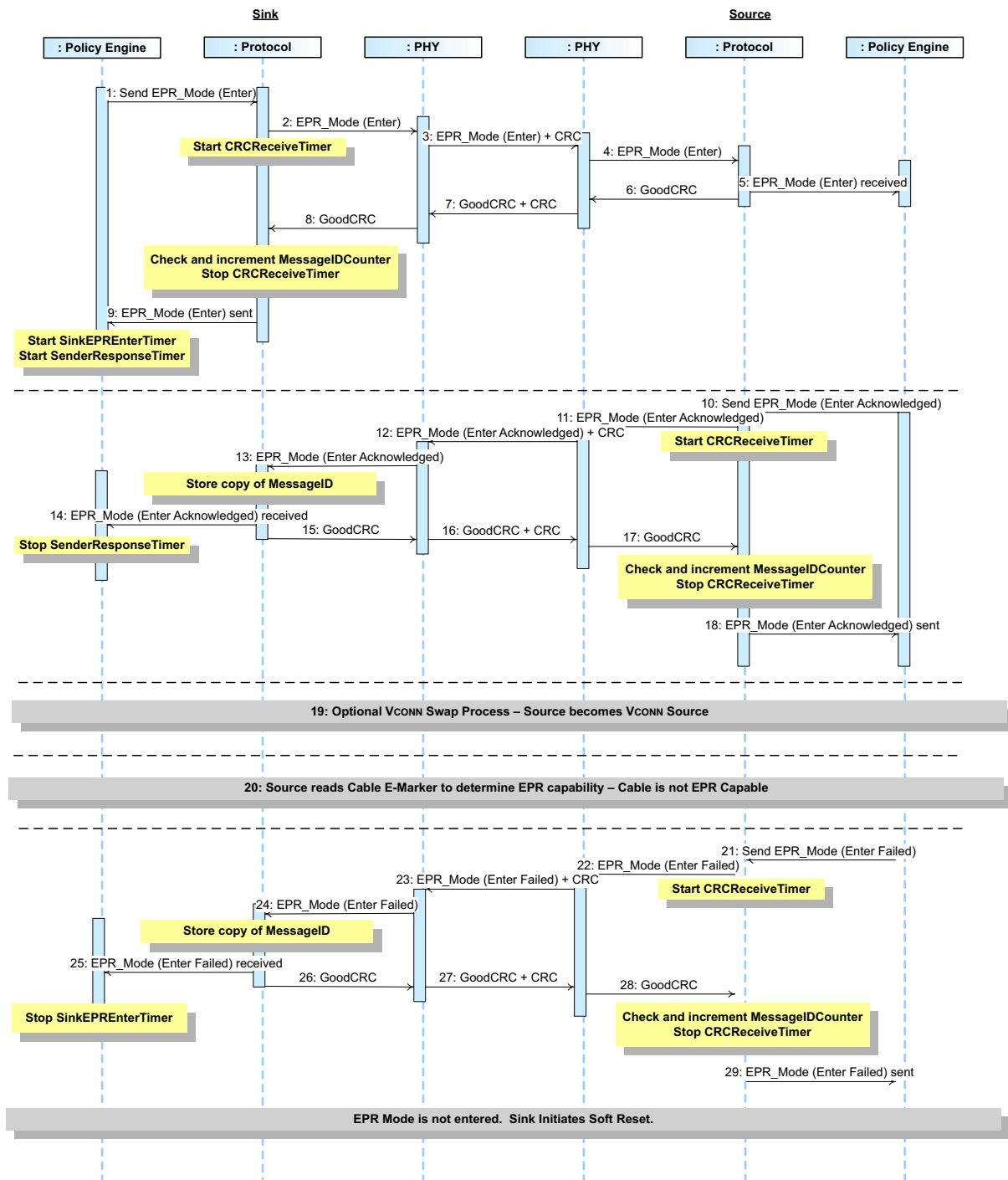
Table 8.39 Steps for Entering EPR Mode (Success)

Step	Sink	Source
16	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Mode (Enter Acknowledged) Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i>
19	If the Source is not the VCONN Source the Source initiates the VCONN Swap process as described in Section 8.3.2.10, "VCONN Swap" .	
20	The Source performs Cable Discovery to determine whether the cable supports EPR. The Cable Discovery process is described in Section 8.3.2.14.1, "Discover Identity" .	
21		<i>The Source is now the VCONN Source and has determined that the Sink and the cable are EPR Capable. The Policy Engine tells the Protocol Layer to form a EPR_Mode (Enter Succeeded) Message.</i>
22		<i>Protocol Layer creates the EPR_Mode (Enter Succeeded) Message and passes to PHY Layer.</i>
23	<i>PHY Layer receives the EPR_Mode (Enter Succeeded) Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the EPR_Mode (Enter Succeeded) Message. Starts CRCReceiveTimer.</i>
24	<i>PHY Layer removes the CRC and forwards the EPR_Mode (Enter Succeeded) Message to the Protocol Layer.</i>	
25	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer passes the EPR_Mode (Enter Succeeded) information to the Policy Engine. The Policy Engine stops the SinkEPREnterTimer.</i>	
26	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
27	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
28		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
29		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Mode (Enter Succeeded) Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i>
EPR Mode Entered		

Entering EPR Mode (Failure due to non-EPR cable)

This is an example of an Enter *EPR Mode* operation where the Sink requests *EPR Mode* when this process fails due to the cable not being capable of EPR. [Figure 8.13, "Entering EPR Mode \(Failure due to non-EPR cable\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter EPR process.

Figure 8.13 Entering EPR Mode (Failure due to non-EPR cable)



[Table 8.40. "Steps for Entering EPR Mode \(Failure due to non-EPR cable\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.13. "Entering EPR Mode \(Failure due to non-EPR cable\)"](#) above.

Table 8.40 Steps for Entering EPR Mode (Failure due to non-EPR cable)

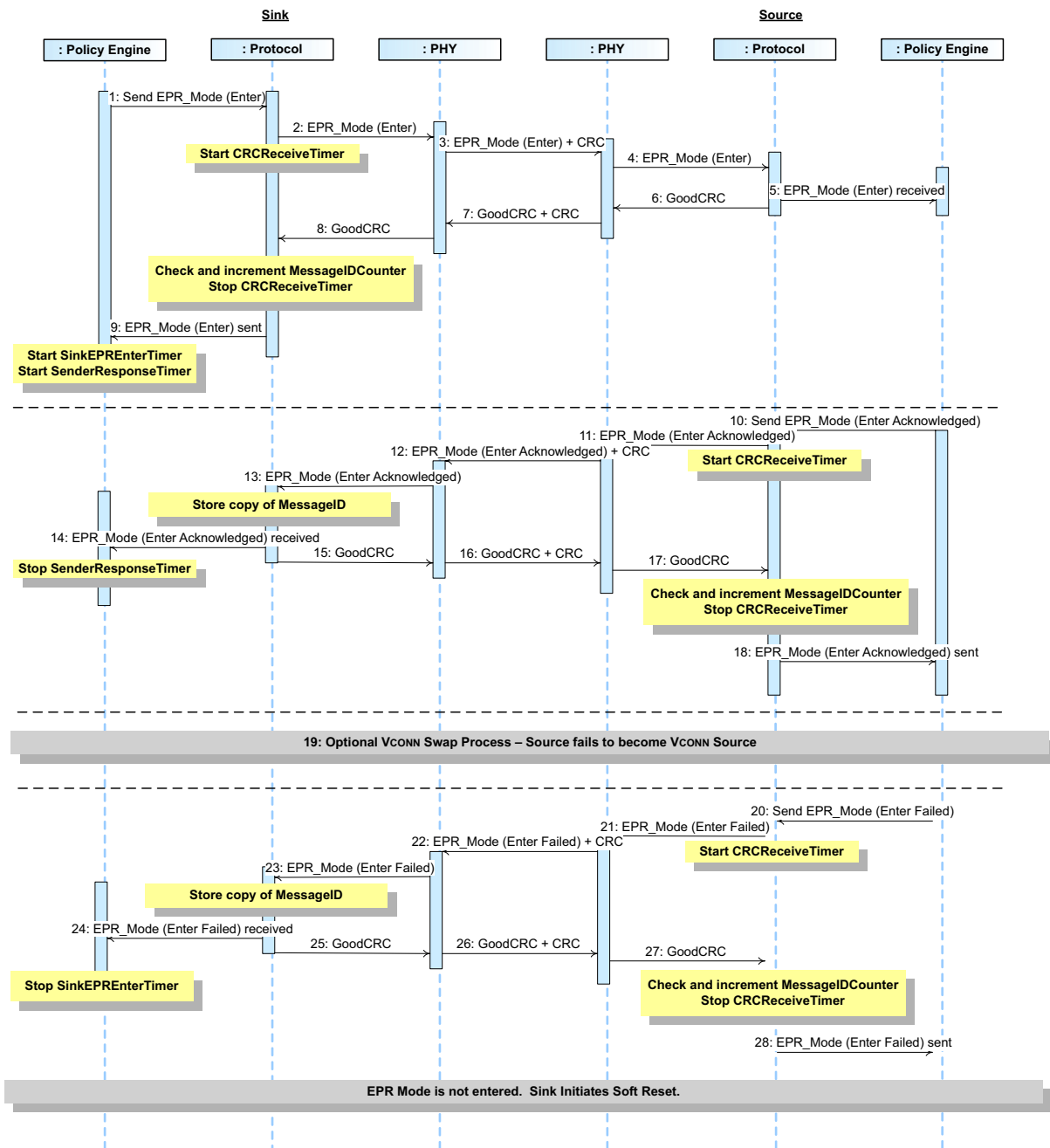
Step	Sink	Source
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an EPR_Mode (Enter) <i>Message</i> to request entry to <i>EPR Mode</i> .	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the EPR_Mode (Enter) <i>Message</i> . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Mode (Enter) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Mode (Enter) <i>Message</i> to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Mode (Enter) <i>Message</i> information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Mode (Enter) <i>Message</i> was successfully sent. The <i>Policy Engine</i> starts the SenderResponseTimer and the SinkEPREnterTimer .	
10		<i>Policy Engine</i> evaluates the EPR_Mode (Enter) <i>Message</i> sent by the <i>Sink</i> . It tells the <i>Protocol Layer</i> to form a EPR_Mode (Enter Acknowledged) <i>Message</i> .
11		<i>Protocol Layer</i> creates the EPR_Mode (Enter Acknowledged) <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the EPR_Mode (Enter Acknowledged) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the EPR_Mode (Enter Acknowledged) <i>Message</i> . Starts CRCReceiveTimer .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Mode (Enter Acknowledged) <i>Message</i> to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the EPR_Mode (Enter Acknowledged) information to the <i>Policy Engine</i> . The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	The <i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it to its <i>PHY Layer</i> .	

Table 8.40 Steps for Entering EPR Mode (Failure due to non-EPR cable)

Step	Sink	Source
16	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Mode (Enter Acknowledged) Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i>
19	If the Source is not the VCONN Source the Source initiates the VCONN Swap process as described in Section 8.3.2.10, "VCONN Swap" .	
20	The Source performs Cable Discovery to determine whether the cable supports EPR; cable is not EPR Capable. The Cable Discovery process is described in Section 8.3.2.14.1, "Discover Identity" .	
21		<p>The Source determines that there has been a failure or incompatibility during the EPR process (see Section 6.4.10, "EPR_Mode Message").</p> <p>The Policy Engine tells the Protocol Layer to form a EPR_Mode (Enter Failed) Message.</p>
22		<i>Protocol Layer creates the EPR_Mode (Enter Failed) Message and passes to PHY Layer.</i>
23	<i>PHY Layer receives the EPR_Mode (Enter Failed) Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the EPR_Mode (Enter Failed) Message. Starts CRCReceiveTimer.</i>
24	<i>PHY Layer removes the CRC and forwards the EPR_Mode (Enter Failed) Message to the Protocol Layer.</i>	
25	<p><i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i></p> <p><i>The Protocol Layer passes the EPR_Mode (Enter Failed) information to the Policy Engine. The Policy Engine stops the SinkEPREnterTimer.</i></p>	
26	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
27	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
28		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
29		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Mode (Enter Failed) Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i>
EPR Mode is not entered. Sink Initiates Soft Reset		

This is an example of an Enter *EPR Mode* operation where the *Sink* requests *EPR Mode* when this process fails due to a failure of the *VCONN Swap* process. [Figure 8.14, "Entering EPR Mode \(Failure of VCONN Swap\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter EPR process.

Figure 8.14 Entering EPR Mode (Failure of VCONN Swap)



[Table 8.41, "Steps for Entering EPR Mode \(Failure of VCONN Swap\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.14, "Entering EPR Mode \(Failure of VCONN Swap\)"](#) above.

Table 8.41 Steps for Entering EPR Mode (Failure of VCONN Swap)

Step	Sink	Source
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an EPR_Mode (Enter) <i>Message</i> to request entry to <i>EPR Mode</i> .	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the EPR_Mode (Enter) <i>Message</i> . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Mode (Enter) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Mode (Enter) <i>Message</i> to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Mode (Enter) <i>Message</i> information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Mode (Enter) <i>Message</i> was successfully sent. The <i>Policy Engine</i> starts the SenderResponseTimer and the SinkEPREnterTimer .	
10		<i>Policy Engine</i> evaluates the EPR_Mode (Enter) <i>Message</i> sent by the <i>Sink</i> . It tells the <i>Protocol Layer</i> to form a EPR_Mode (Enter Acknowledged) <i>Message</i> .
11		<i>Protocol Layer</i> creates the EPR_Mode (Enter Acknowledged) <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the EPR_Mode (Enter Acknowledged) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the EPR_Mode (Enter Acknowledged) <i>Message</i> . Starts CRCReceiveTimer .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Mode (Enter Acknowledged) <i>Message</i> to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the EPR_Mode (Enter Acknowledged) information to the <i>Policy Engine</i> . The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	The <i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it to its <i>PHY Layer</i> .	

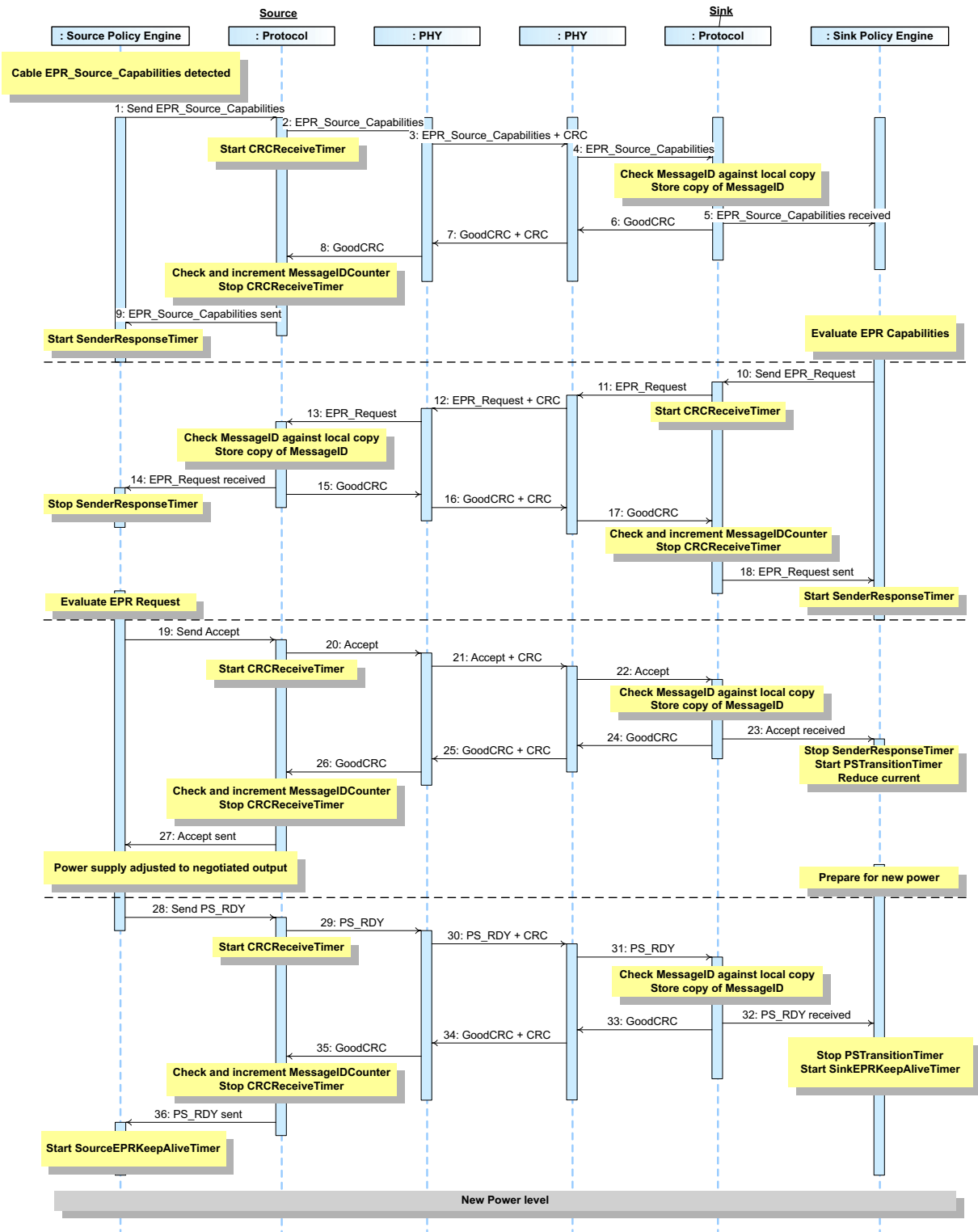
Table 8.41 Steps for Entering EPR Mode (Failure of VCONN Swap)

Step	Sink	Source
16	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Mode (Enter Acknowledged) Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i>
19	If the Source is not the VCONN Source the Source initiates the VCONN Swap process as described in Section 8.3.2.10, "VCONN Swap" . In this case the VCONN Swap process fails.	
20		<i>The Source determines that there has been a failure or incompatibility during the EPR process (see Section 6.4.10, "EPR_Mode Message"). The Policy Engine tells the Protocol Layer to form a EPR_Mode (Enter Failed) Message.</i>
21		<i>Protocol Layer creates the EPR_Mode (Enter Failed) Message and passes to PHY Layer.</i>
22	<i>PHY Layer receives the EPR_Mode (Enter Failed) Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the EPR_Mode (Enter Failed) Message. Starts CRCReceiveTimer.</i>
23	<i>PHY Layer removes the CRC and forwards the EPR_Mode (Enter Failed) Message to the Protocol Layer.</i>	
24	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer passes the EPR_Mode (Enter Failed) information to the Policy Engine. The Policy Engine stops the SinkEPREnterTimer.</i>	
25	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
26	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
27		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
28		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Mode (Enter Failed) Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i>
<i>EPR Mode is not entered. Sink Initiates Soft Reset</i>		

[*Figure 8.15, "Successful Fixed EPR Power Negotiation"*](#) illustrates an example of a successful *Message* flow while negotiating an *Explicit Contract* in *EPR Mode*. The *Negotiation* goes through several distinct phases:

- The *Source* sends out its power *Capabilities* in an *EPR_Source_Capabilities* *Message*.
- The *Sink* evaluates these *Capabilities* and, in the request phase, selects one power level by sending an *EPR_Request* *Message*.
- The *Source* evaluates the request and accepts the request with an *Accept* *Message*.
- The *Source* transitions to the new power level and then informs the *Sink* by sending a *PS_RDY* *Message*.
- The *Sink* starts using the new power level.
- the *Source* starts its keep alive timer
- the *Sink* starts its request timer to send periodic *EPR_KeepAlive* *Messages*

Figure 8.15 Successful Fixed EPR Power Negotiation



[Table 8.42, "Steps for a successful EPR Power Negotiation"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.15, "Successful Fixed EPR Power Negotiation"](#) above.

Table 8.42 Steps for a successful EPR Power Negotiation

Step	Source	Sink
1	The <i>Cable Capabilities</i> are detected if these are not already known (see Section 4.4, "Cable Type Detection"). <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a EPR_Source_Capabilities Message that represents the power supply's present capabilities.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Source_Capabilities Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Source_Capabilities Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Source_Capabilities Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Source_Capabilities Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Source_Capabilities Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the EPR_Source_Capabilities Message sent by the <i>Source</i> and selects which power it would like. It tells the <i>Protocol Layer</i> to form the data (e.g., <i>Power Data Object</i>) that represents its Request into a Message.
11		<i>Protocol Layer</i> creates the EPR_Request Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the EPR_Request Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the EPR_Request Message. Starts CRCReceiveTimer .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the EPR_Request Message to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops SenderResponseTimer .	
15	The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .	

Table 8.42 Steps for a successful EPR Power Negotiation

Step	Source	Sink
16	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> .	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .
18		The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . It informs the <i>Policy Engine</i> that the EPR_Request <i>Message</i> was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer . The <i>Policy Engine</i> starts SenderResponseTimer .
19	<i>Policy Engine</i> evaluates the EPR_Request <i>Message</i> sent by the <i>Sink</i> and decides if it can meet the request. It tells the <i>Protocol Layer</i> to form an Accept <i>Message</i> .	
20	The <i>Protocol Layer</i> forms the Accept <i>Message</i> that is passed to the <i>PHY Layer</i> .	
21	<i>PHY Layer</i> appends <i>CRC</i> and sends the Accept <i>Message</i> . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
22		<i>PHY Layer</i> forwards the Accept <i>Message</i> to the <i>Protocol Layer</i> .
23		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept <i>Message</i> has been received. The <i>Policy Engine</i> stops SenderResponseTimer , starts the PSTransitionTimer and reduces its current draw. The <i>DPM</i> prepares the Power supply for transition to the new power level.
24		The <i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it to its <i>PHY Layer</i> .
25	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> .
26	<i>PHY Layer</i> forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
27	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept <i>Message</i> was successfully sent.	
Power supply Adjusts its Output to the <i>Negotiated</i> Value		
28	The <i>DPM</i> informs the <i>Policy Engine</i> that the power supply has settled at the new operating condition and tells the <i>Protocol Layer</i> to send a PS_RDY <i>Message</i> .	
29	The <i>Protocol Layer</i> forms the PS_RDY <i>Message</i> .	
30	<i>PHY Layer</i> appends <i>CRC</i> and sends the PS_RDY <i>Message</i> . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PS_RDY <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
31		<i>PHY Layer</i> forwards the PS_RDY <i>Message</i> to the <i>Protocol Layer</i> .

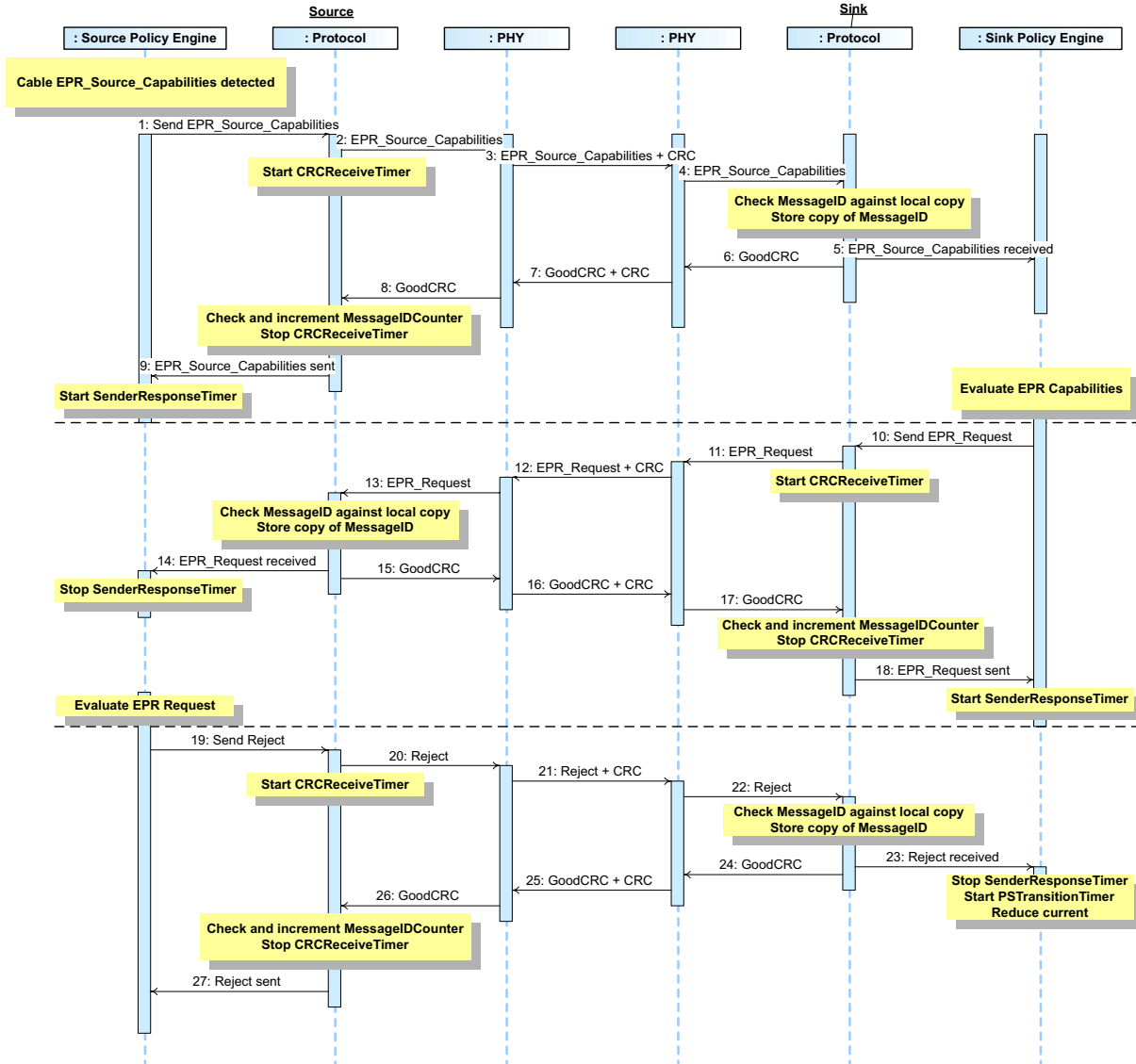
Table 8.42 Steps for a successful EPR Power Negotiation

Step	Source	Sink
32		<p><i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that a RS_RDY has been received. The <i>Policy Engine</i> stops the PSTransitionTimer.</p> <p>The <i>Policy Engine</i> starts the SinkEPRKeepAliveTimer.</p>
33		The <i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it to its <i>PHY Layer</i> .
34	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> .
35	<i>PHY Layer</i> forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . Stops the CRCReceiveTimer .	
36	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY <i>Message</i> was successfully sent.	
37	When in EPR operation the <i>Policy Engine</i> starts the SourceEPRKeepAliveTimer .	

Figure 8.16, "Rejected Fixed EPR Power Negotiation" illustrates an example of a Message flow where the request is rejected while negotiating an Explicit Contract in EPR Mode. The Negotiation goes through several distinct phases:

- The Source sends out its power Capabilities in an *EPR_Source_Capabilities* Message.
- The Sink evaluates these Capabilities and, in the request phase, selects one power level by sending an *EPR_Request* Message.
- The Source evaluates the request and accepts the request with a *Reject* Message.

Figure 8.16 Rejected Fixed EPR Power Negotiation



[Table 8.43, "Steps for a Rejected EPR Power Negotiation"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.16, "Rejected Fixed EPR Power Negotiation"](#) above.

Table 8.43 Steps for a Rejected EPR Power Negotiation

Step	Source	Sink
1	The <i>Cable Capabilities</i> are detected if these are not already known (see Section 4.4, "Cable Type Detection"). <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>EPR_Source_Capabilities</i> Message that represents the power supply's present capabilities.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Source_Capabilities</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>EPR_Source_Capabilities</i> Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>EPR_Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>EPR_Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> evaluates the <i>EPR_Source_Capabilities</i> Message sent by the <i>Source</i> and selects which power it would like. It tells the <i>Protocol Layer</i> to form the data (e.g., <i>Power Data Object</i>) that represents its Request into a Message.
11		<i>Protocol Layer</i> creates the <i>EPR_Request</i> Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>EPR_Request</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>EPR_Request</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_Request</i> Message to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops <i>SenderResponseTimer</i> .	
15	The <i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it to its <i>PHY Layer</i> .	

Table 8.43 Steps for a Rejected EPR Power Negotiation

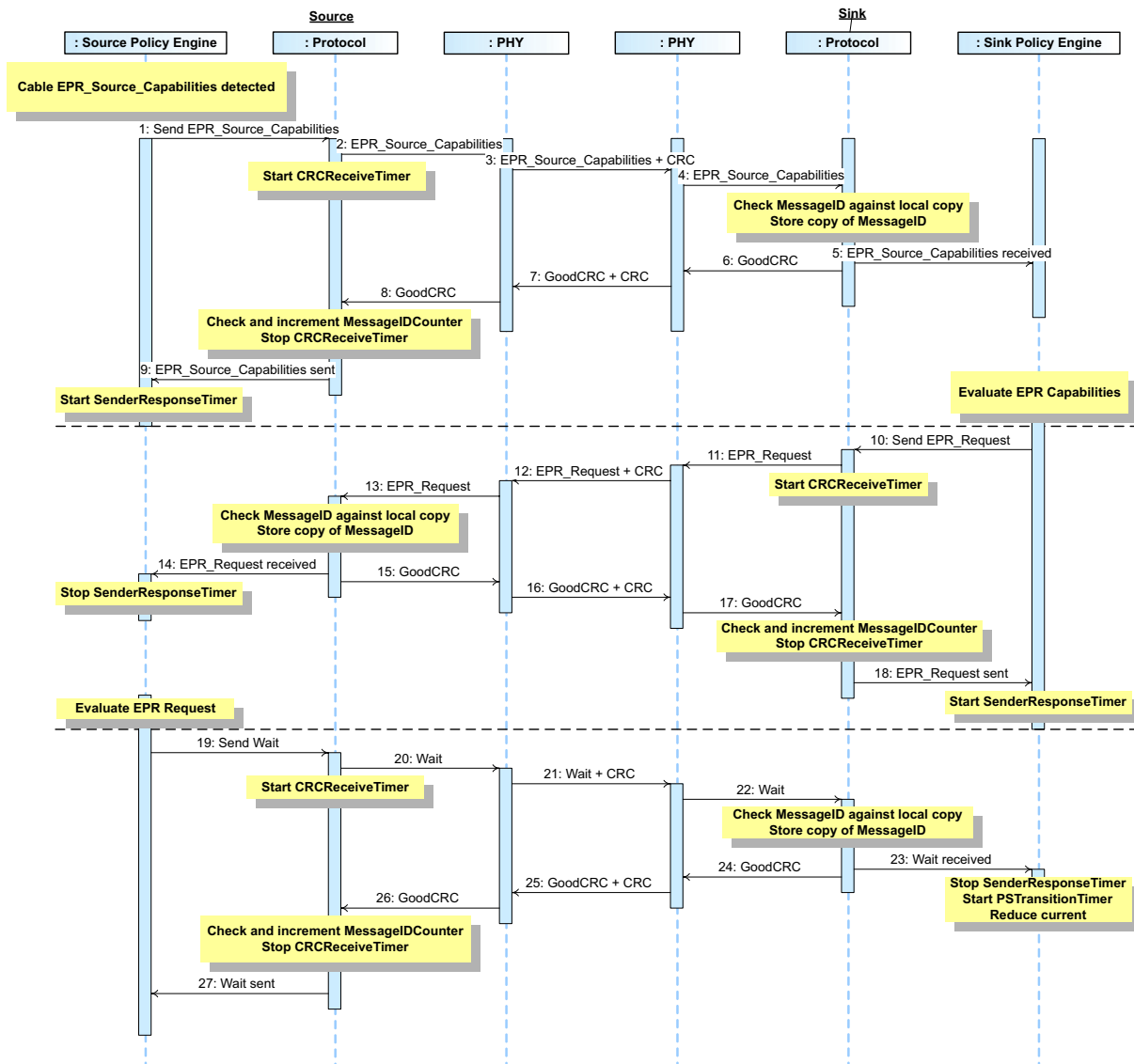
Step	Source	Sink
16	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer. The Policy Engine starts SenderResponseTimer.</i>
19	<i>Policy Engine evaluates the EPR_Request Message sent by the Sink and decides it can't meet the request. It tells the Protocol Layer to form a Reject Message.</i>	
20	<i>The Protocol Layer forms the Reject Message that is passed to the PHY Layer.</i>	
21	<i>PHY Layer appends CRC and sends the Reject Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
22		<i>PHY Layer forwards the Reject Message to the Protocol Layer.</i>
23		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. Protocol Layer informs the Policy Engine that a Reject Message has been received. The Policy Engine stops SenderResponseTimer.</i>
24		<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>
25	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends CRC and sends the Message.</i>
26	<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer. The Protocol Layer verifies and increments the MessageIDCounter and stops the CRCReceiveTimer.</i>	
27	<i>The Protocol Layer informs the Policy Engine that a Reject Message was successfully sent.</i>	

EPR Explicit Contract Negotiation (Wait)

Figure 8.17, "Wait response to Fixed EPR Power Negotiation" illustrates an example of a *Message* flow where the request is responded to with wait while negotiating an *Explicit Contract* in *EPR Mode*. The *Negotiation* goes through several distinct phases:

- The *Source* sends out its power *Capabilities* in an *EPR_Source_Capabilities* Message.
- The *Sink* evaluates these *Capabilities* and, in the request phase, selects one power level by sending an *EPR_Request* Message.
- The *Source* evaluates the request and accepts the request with a *Wait* Message.

Figure 8.17 Wait response to Fixed EPR Power Negotiation



[Table 8.44, "Steps for a Wait response to an EPR Power Negotiation"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.17, "Wait response to Fixed EPR Power Negotiation"](#) above.

Table 8.44 Steps for a Wait response to an EPR Power Negotiation

Step	Source	Sink
1	The <i>Cable Capabilities</i> are detected if these are not already known (see Section 4.4, "Cable Type Detection"). <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>EPR_Source_Capabilities</i> Message that represents the power supply's present capabilities.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Source_Capabilities</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>EPR_Source_Capabilities</i> Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>EPR_Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>EPR_Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> evaluates the <i>EPR_Source_Capabilities</i> Message sent by the <i>Source</i> and selects which power it would like. It tells the <i>Protocol Layer</i> to form the data (e.g., <i>Power Data Object</i>) that represents its Request into a Message.
11		<i>Protocol Layer</i> creates the <i>EPR_Request</i> Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>EPR_Request</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>EPR_Request</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_Request</i> Message to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops <i>SenderResponseTimer</i> .	
15	The <i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it to its <i>PHY Layer</i> .	

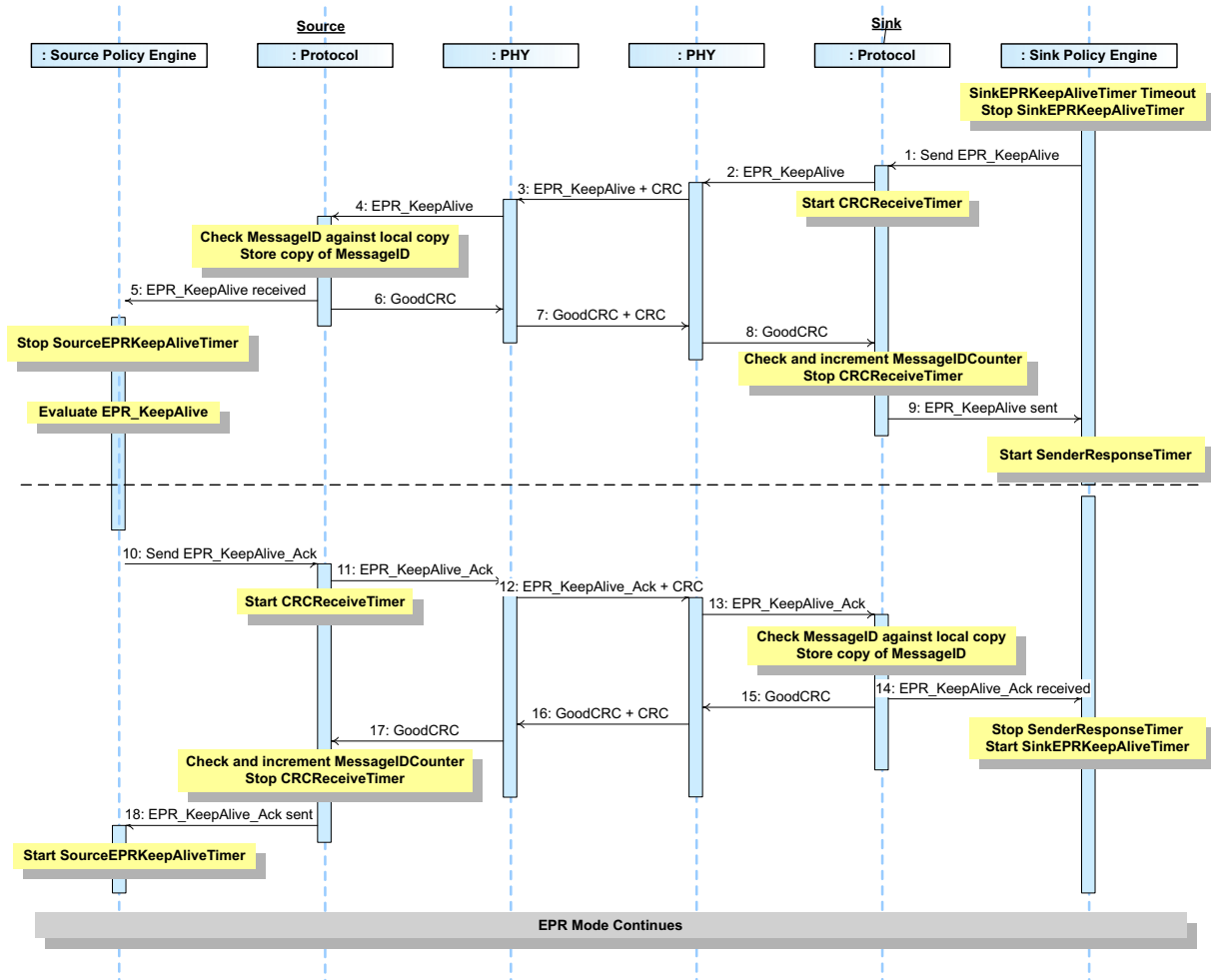
Table 8.44 Steps for a Wait response to an EPR Power Negotiation

Step	Source	Sink
16	<i>PHY Layer appends CRC and sends the Message.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer. The Policy Engine starts SenderResponseTimer.</i>
19	<i>Policy Engine evaluates the EPR_Request Message sent by the Sink and decides if it can meet the request. It tells the Protocol Layer to form a Wait Message.</i>	
20	<i>The Protocol Layer forms the Wait Message that is passed to the PHY Layer.</i>	
21	<i>PHY Layer appends CRC and sends the Wait Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>
22		<i>PHY Layer forwards the Wait Message to the Protocol Layer.</i>
23		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. Protocol Layer informs the Policy Engine that a Wait Message has been received. The Policy Engine stops SenderResponseTimer.</i>
24		<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>
25	<i>PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends CRC and sends the Message.</i>
26	<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer. The Protocol Layer verifies and increments the MessageIDCounter and stops the CRCReceiveTimer.</i>	
27	<i>The Protocol Layer informs the Policy Engine that a Wait Message was successfully sent.</i>	

EPR Keep Alive

This is an example of keep alive operation during an *Explicit Contract* in *EPR Mode*. [Figure 8.18, "EPR Keep Alive"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.18 EPR Keep Alive



[Table 8.45, "Steps for EPR Keep Alive"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.18, "EPR Keep Alive"](#) above.

Table 8.45 Steps for EPR Keep Alive

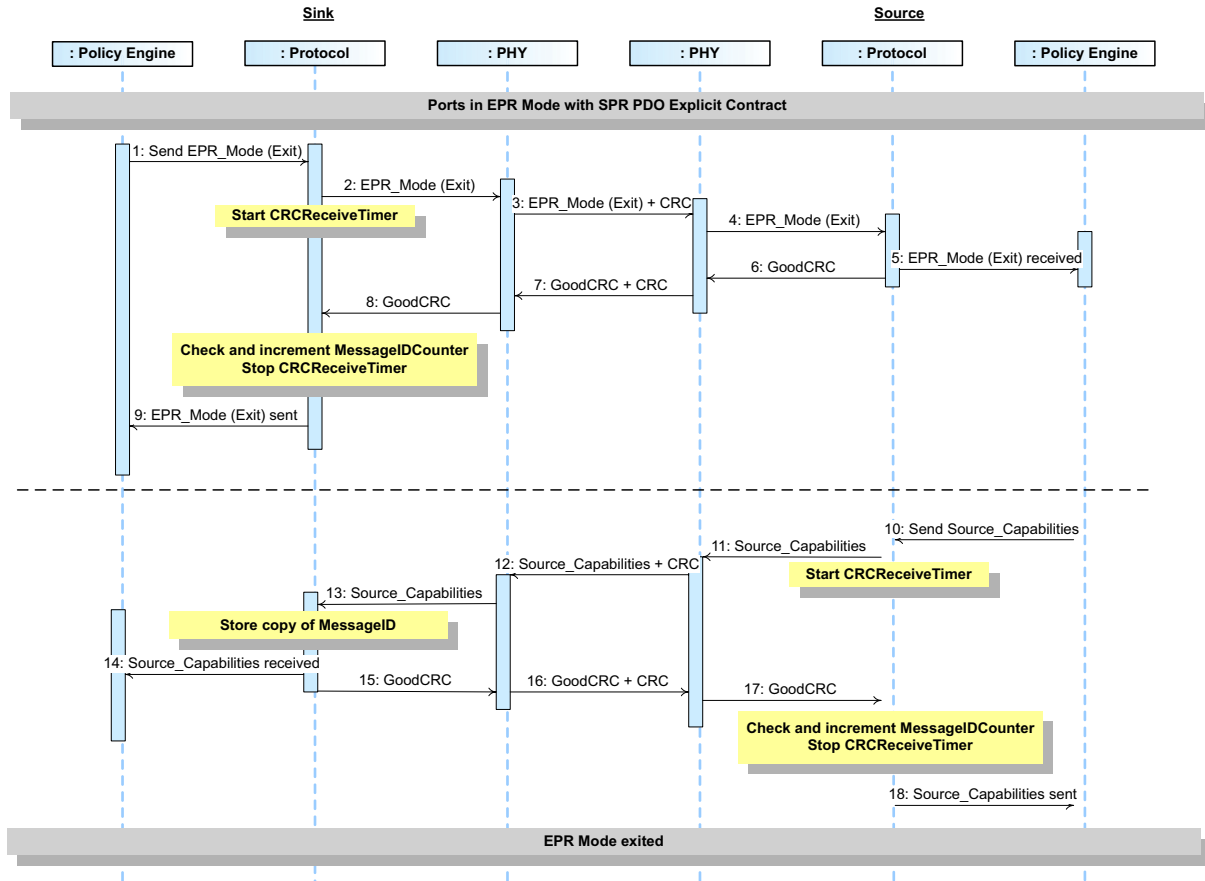
Step	Source	Sink
1		The <i>SinkEPRKeepAliveTimer</i> times out in the <i>Policy Engine</i> . The <i>Policy Engine</i> stops the <i>SinkEPRKeepAliveTimer</i> timer and tells the <i>Protocol Layer</i> to form an <i>EPR_KeepAlive</i> Message.
2		The <i>Protocol Layer</i> creates the <i>EPR_KeepAlive</i> Message and passes it to <i>PHY Layer</i> . The <i>Protocol Layer</i> .
3	<i>PHY Layer</i> receives the <i>EPR_KeepAlive</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Request</i> Message. Starts <i>CRCReceiveTimer</i> .
4	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_KeepAlive</i> Message to the <i>Protocol Layer</i> .	
5	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the Request information to the <i>Policy Engine</i> . <i>Policy Engine</i> stops the <i>SourceEPRKeepAliveTimer</i> .	
6	The <i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it to its <i>PHY Layer</i> .	
7	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
8		<i>PHY Layer</i> forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
9		The <i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> . It informs the <i>Policy Engine</i> that the <i>SinkEPRKeepAliveTimer</i> Message was successfully sent. The <i>Protocol Layer</i> stops the <i>CRCReceiveTimer</i> . The <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .
10	<i>Policy Engine</i> requests the <i>DPM</i> to evaluate the <i>SourceEPRKeepAliveTimer</i> Message sent by the Sink and decides if the Source can meet the request. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>EPR_KeepAlive_Ack</i> Message.	
11	The <i>Protocol Layer</i> forms the <i>EPR_KeepAlive_Ack</i> Message that is passed to the <i>PHY Layer</i> .	
12	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>EPR_KeepAlive_Ack</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>EPR_KeepAlive_Ack</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
13		<i>PHY Layer</i> forwards the <i>EPR_KeepAlive_Ack</i> Message to the <i>Protocol Layer</i> .
14		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an <i>Accept</i> Message has been received. The <i>Policy Engine</i> stops <i>SenderResponseTimer</i> , starts the <i>SinkEPRKeepAliveTimer</i> .

Table 8.45 Steps for EPR Keep Alive

Step	Source	Sink
15		The <i>Protocol Layer</i> generates a <i>GoodCRC Message</i> and passes it to its <i>PHY Layer</i> .
16	<i>PHY Layer</i> receives the <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC Message</i> .
17	<i>PHY Layer</i> forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops the <i>CRCReceiveTimer</i> .	
18	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an <i>EPR_KeepAlive_Ack Message</i> was successfully sent. The <i>Policy Engine</i> starts the <i>SourceEPRKeepAliveTimer</i> .	
EPR Mode Continues		

This is an example of an Exit *EPR Mode* operation where the *Sink* requests *EPR Mode* to be exited. [Figure 8.19, "Exiting EPR Mode \(Sink Initiated\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Exit EPR process.

Figure 8.19 Exiting EPR Mode (Sink Initiated)



[Table 8.46, "Steps for Exiting EPR Mode \(Sink Initiated\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.19, "Exiting EPR Mode \(Sink Initiated\)"](#) above.

Table 8.46 Steps for Exiting EPR Mode (Sink Initiated)

Step	Sink	Source
The Port Partners are in an <i>Explicit Contract</i> using an <i>SPR (A)PDO</i> (Voltage <= 20V)		
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an <i>EPR_Mode</i> (Exit) <i>Message</i> to request entry to <i>EPR Mode</i> .	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>EPR_Mode</i> (Exit) <i>Message</i> . Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>EPR_Mode</i> (Exit) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_Mode</i> (Exit) <i>Message</i> to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>EPR_Mode</i> (Exit) <i>Message</i> information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>EPR_Mode</i> (Exit) <i>Message</i> was successfully sent.	
10		<i>Policy Engine</i> evaluates the <i>EPR_Mode</i> (Exit) <i>Message</i> sent by the Sink. It tells the <i>Protocol Layer</i> to form a <i>Source_Capabilities</i> <i>Message</i> .
11		<i>Protocol Layer</i> creates the <i>Source_Capabilities</i> <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Source_Capabilities</i> <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Source_Capabilities</i> <i>Message</i> . Starts <i>CRCReceiveTimer</i> .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Source_Capabilities</i> <i>Message</i> to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the <i>EPR_Mode</i> (Enter Succeeded) information to the <i>Policy Engine</i> .	
15	The <i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it to its <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> .	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

Table 8.46 Steps for Exiting EPR Mode (Sink Initiated)

Step	Sink	Source
17		<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . It informs the <i>Policy Engine</i> that the Source_Capabilities Message was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer .
EPR Mode Exited. Power Negotiation proceeds as defined in Section 8.3.2.2.1.1, "SPR Explicit Contract Negotiation" .		

Exiting EPR Mode (Source Initiated)

[Table 8.47, "Steps for Exiting EPR Mode \(Source Initiated\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.20, "Exiting EPR Mode \(Source Initiated\)"](#) above.

Table 8.47 Steps for Exiting EPR Mode (Source Initiated)

Step	Sink	Source
The Port Partners are in an <i>Explicit Contract</i> using an <i>SPR (A)PDO</i> (Voltage <= 20V)		
1		The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an <i>EPR_Mode</i> (Exit) <i>Message</i> to request entry to <i>EPR Mode</i> .
2		<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .
3	<i>PHY Layer</i> receives the <i>EPR_Mode</i> (Exit) <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>EPR_Mode</i> (Exit) <i>Message</i> . Starts <i>CRCReceiveTimer</i> .
4	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>EPR_Mode</i> (Exit) <i>Message</i> to the <i>Protocol Layer</i> .	
5	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>EPR_Mode</i> (Exit) <i>Message</i> information to the <i>Policy Engine</i> that consumes it.	
6	<i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it <i>PHY Layer</i> .	
7	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> <i>Message</i> .	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the <i>CRC</i> to verify the <i>Message</i> .
8		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> <i>Message</i> to the <i>Protocol Layer</i> .
9		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>EPR_Mode</i> (Exit) <i>Message</i> was successfully sent.
10		<i>Policy Engine</i> evaluates the <i>EPR_Mode</i> (Exit) <i>Message</i> sent by the <i>Sink</i> . It tells the <i>Protocol Layer</i> to form a <i>Source_Capabilities</i> <i>Message</i> .
11		<i>Protocol Layer</i> creates the <i>Source_Capabilities</i> <i>Message</i> and passes to <i>PHY Layer</i> . Starts <i>CRCReceiveTimer</i> .
12	<i>PHY Layer</i> receives the <i>Source_Capabilities</i> <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Source_Capabilities</i> <i>Message</i> .
13	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Source_Capabilities</i> <i>Message</i> to the <i>Protocol Layer</i> .	
14	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> passes the <i>EPR_Mode</i> (Enter Succeeded) information to the <i>Policy Engine</i> .	
15	The <i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it to its <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> .	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

Table 8.47 Steps for Exiting EPR Mode (Source Initiated)

Step	Sink	Source
17		<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . It informs the <i>Policy Engine</i> that the Source_Capabilities Message was successfully sent. The <i>Protocol Layer</i> stops the CRCReceiveTimer .
EPR Mode Exited. Power Negotiation proceeds as defined in Section 8.3.2.2.1.1, "SPR Explicit Contract Negotiation" .		

8.3.2.2.5

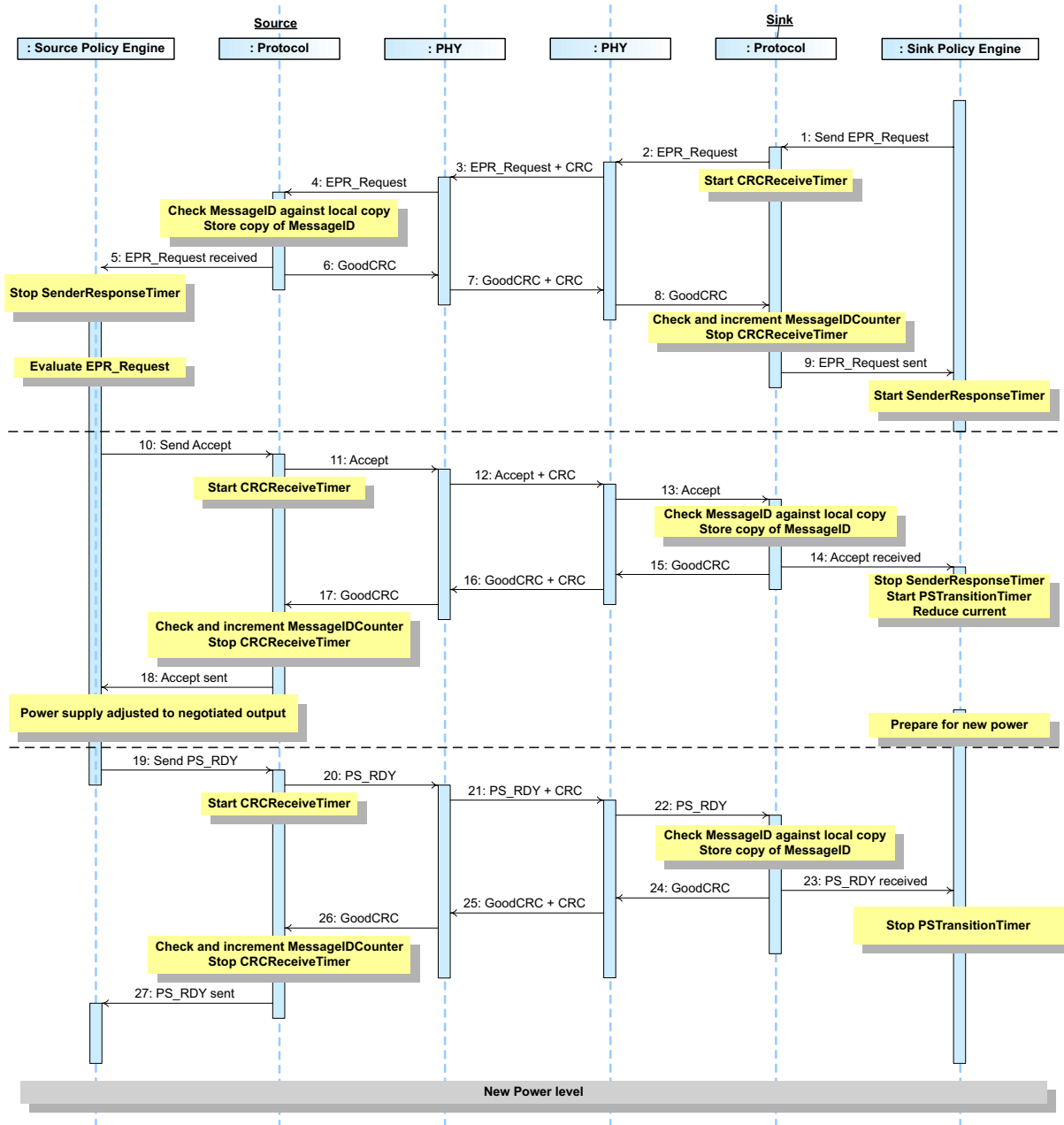
EPR Sink Makes Request

8.3.2.2.5.1

EPR Sink Makes Request (Accept)

This is an example of EPR when a *Sink* makes a Request which is Accepted during an *Explicit Contract*. [Figure 8.21, "EPR Sink Makes Request \(Accept\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.21 EPR Sink Makes Request (Accept)



[Table 8.48, "Steps for EPR Sink Makes Request \(Accept\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.21, "EPR Sink Makes Request \(Accept\)"](#) above.

Table 8.48 Steps for EPR Sink Makes Request (Accept)

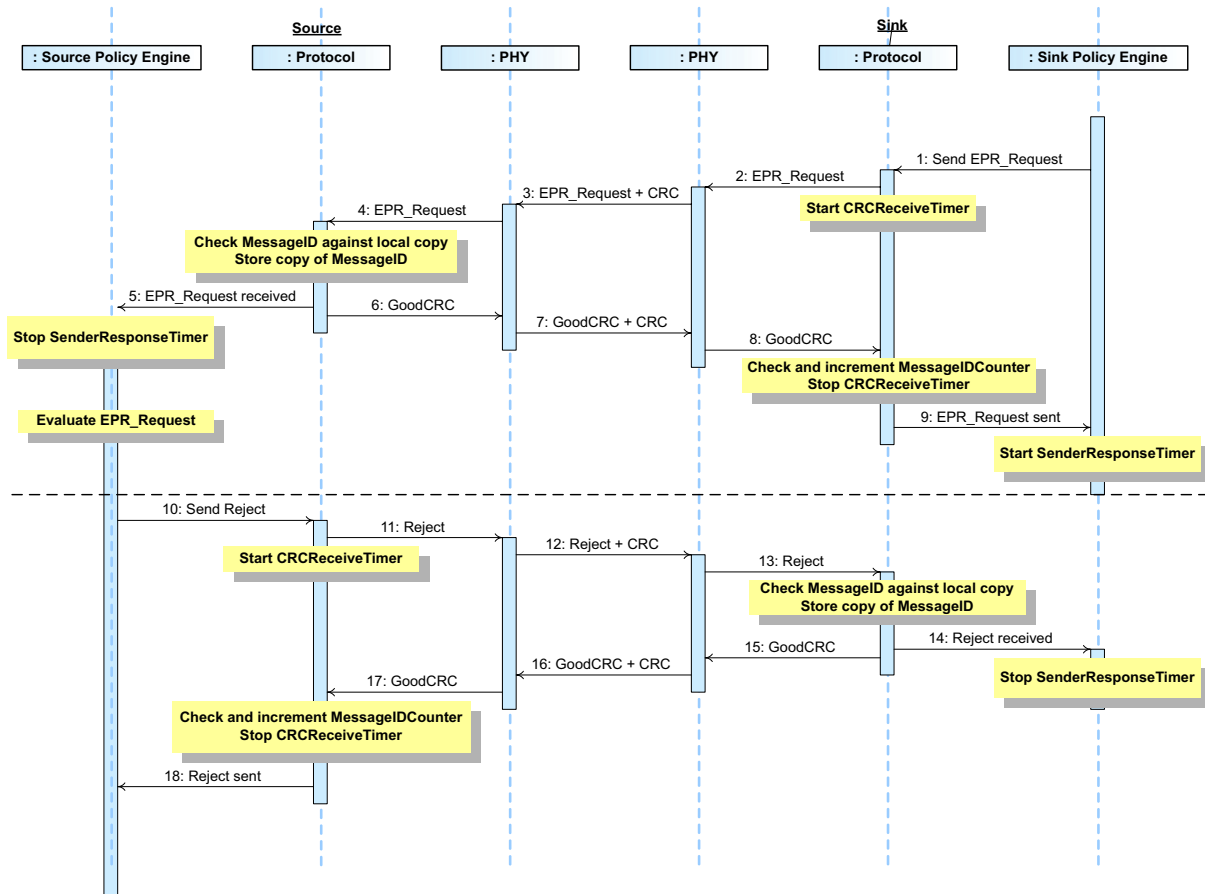
Step	Source	Sink
1		<i>DPM tells the Policy Engine to request a different power level. The Policy Engine tells the Protocol Layer to form an EPR_Request Message.</i> <i>The Protocol Layer creates the EPR_Request Message and passes it to PHY Layer.</i>
2	<i>PHY Layer receives the EPR_Request Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the EPR_Request Message. Starts CRCReceiveTimer.</i>
3	<i>PHY Layer removes the CRC and forwards the EPR_Request Message to the Protocol Layer.</i>	
4	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>The Protocol Layer passes the Request information to the Policy Engine.</i>	
5	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
6	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
7		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
8		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i> <i>The Policy Engine starts SenderResponseTimer.</i>
9	<i>Policy Engine requests the DPM to evaluate the EPR_Request Message sent by the Sink and decides if the Source can meet the request. The Policy Engine tells the Protocol Layer to form an Accept Message.</i>	
10	<i>The Protocol Layer forms the Accept Message that is passed to the PHY Layer.</i>	
11	<i>PHY Layer appends CRC and sends the Accept Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Accept Message and compares the CRC it calculated with the one sent to verify the Message.</i>
12		<i>PHY Layer forwards the Accept Message to the Protocol Layer.</i>
13		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>Protocol Layer informs the Policy Engine that an Accept Message has been received. The Policy Engine stops SenderResponseTimer, starts the PSTransitionTimer and reduces its current draw.</i> <i>The DPM prepares the Power supply for transition to the new power level.</i>

Table 8.48 Steps for EPR Sink Makes Request (Accept)

Step	Source	Sink
14		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that an Accept Message was successfully sent.	
Power supply Adjusts its Output to the <i>Negotiated</i> Value		
18	The <i>DPM</i> informs the <i>Policy Engine</i> that the power supply has settled at the new operating condition and tells the <i>Protocol Layer</i> to send a PS_RDY Message.	
19	The <i>Protocol Layer</i> forms the PS_RDY Message.	
20	<i>PHY Layer</i> appends <i>CRC</i> and sends the PS_RDY Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PS_RDY Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
21		<i>PHY Layer</i> forwards the PS_RDY Message to the <i>Protocol Layer</i> .
22		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a PS_RDY has been received. The <i>Policy Engine</i> stops the PSTransitionTimer .
23		The <i>Protocol Layer</i> generates a GoodCRC Message and passes it to its <i>PHY Layer</i> .
24	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
25	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter . Stops the CRCReceiveTimer .	
26	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent.	
New Power Level <i>Negotiated</i>		

This is an example of EPR when a *Sink* makes a Request which is Rejected during an *Explicit Contract*. [Figure 8.22, "EPR Sink Makes Request \(Reject\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.22 EPR Sink Makes Request (Reject)



[Table 8.49, "Steps for EPR Sink Makes Request \(Reject\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.22, "EPR Sink Makes Request \(Reject\)"](#) above.

Table 8.49 Steps for EPR Sink Makes Request (Reject)

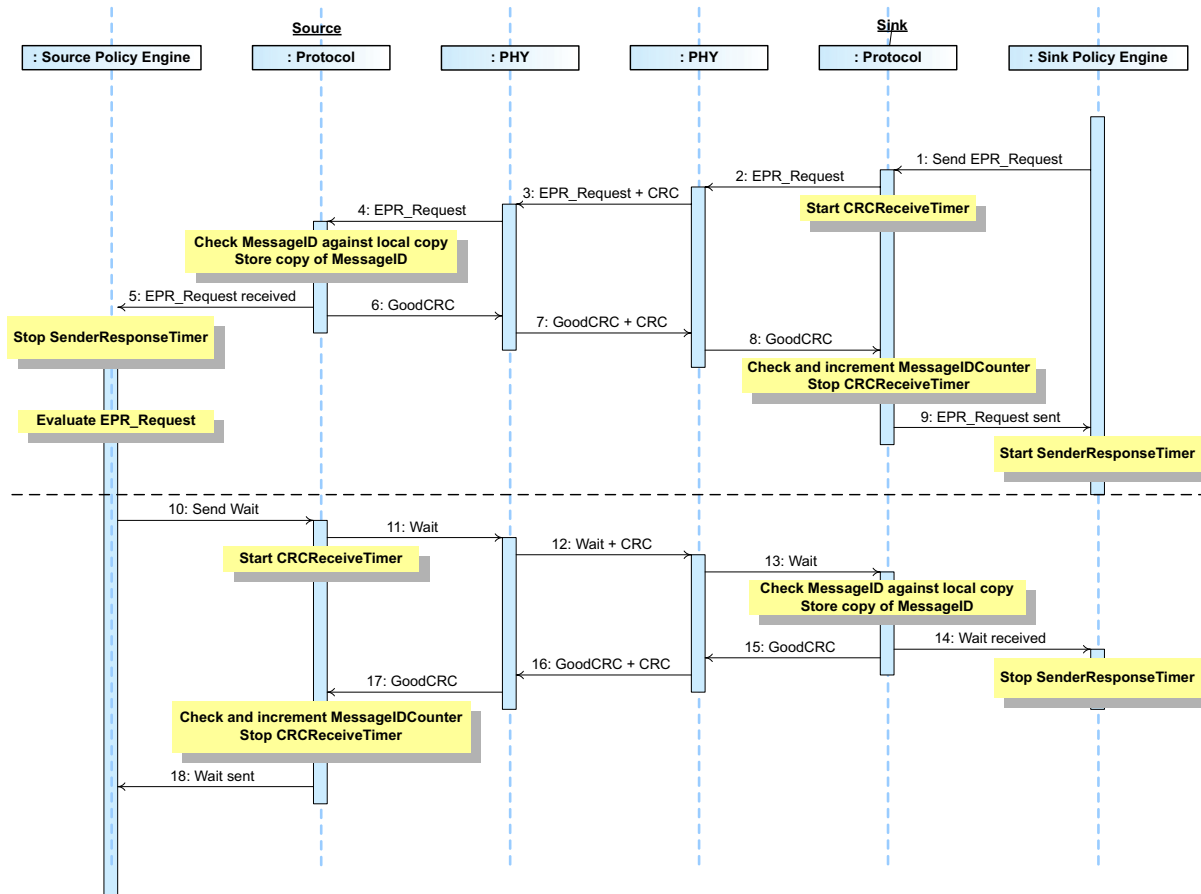
Step	Source	Sink
1		<i>DPM tells the Policy Engine to request a different power level. The Policy Engine tells the Protocol Layer to form an EPR_Request Message.</i> <i>The Protocol Layer creates the Request Message and passes it to PHY Layer.</i>
2	<i>PHY Layer receives the EPR_Request Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the EPR_Request Message. Starts CRCReceiveTimer.</i>
3	<i>PHY Layer removes the CRC and forwards the EPR_Request Message to the Protocol Layer.</i>	
4	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>The Protocol Layer passes the Request information to the Policy Engine.</i>	
5	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
6	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
7		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
8		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the EPR_Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i> <i>The Policy Engine starts SenderResponseTimer.</i>
9	<i>Policy Engine requests the DPM to evaluate the EPR_Request Message sent by the Sink and decides that the Source can't meet the request. The Policy Engine tells the Protocol Layer to form a Reject Message.</i>	
10	<i>The Protocol Layer forms the Reject Message that is passed to the PHY Layer.</i>	
11	<i>PHY Layer appends CRC and sends the Reject Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Reject Message and compares the CRC it calculated with the one sent to verify the Message.</i>
12		<i>PHY Layer forwards the Reject Message to the Protocol Layer.</i>
13		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>Protocol Layer informs the Policy Engine that an Reject Message has been received. The Policy Engine informs the DPM that the Request has been rejected.</i>
14		<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>

Table 8.49 Steps for EPR Sink Makes Request (Reject)

Step	Source	Sink
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Reject Message was successfully sent.	

This is an example of SPR when a *Sink* makes a Request which is responded to with a *Wait Message* during an *Explicit Contract*. [Figure 8.23, "EPR Sink Makes Request \(Wait\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the keep alive.

Figure 8.23 EPR Sink Makes Request (Wait)



[Table 8.50, "Steps for SPR Sink Makes Request \(Wait\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.23, "EPR Sink Makes Request \(Wait\)"](#) above.

Table 8.50 Steps for SPR Sink Makes Request (Wait)

Step	Source	Sink
1		<i>DPM tells the Policy Engine to request a different power level. The Policy Engine tells the Protocol Layer to form an EPR_Request Message.</i> <i>The Protocol Layer creates the EPR_Request Message and passes it to PHY Layer.</i>
2	<i>PHY Layer receives the EPR_Request Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the EPR_Request Message. Starts CRCReceiveTimer.</i>
3	<i>PHY Layer removes the CRC and forwards the EPR_Request Message to the Protocol Layer.</i>	
4	<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>The Protocol Layer passes the Request information to the Policy Engine.</i>	
5	<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>	
6	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives the GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
7		<i>PHY Layer forwards the GoodCRC Message to the Protocol Layer.</i>
8		<i>The Protocol Layer verifies and increments the MessageIDCounter. It informs the Policy Engine that the Request Message was successfully sent. The Protocol Layer stops the CRCReceiveTimer.</i> <i>The Policy Engine starts SenderResponseTimer.</i>
9	<i>Policy Engine requests the DPM to evaluate the EPR_Request Message sent by the Sink and decides if the Source can meet the request. The Policy Engine tells the Protocol Layer to form a Wait Message.</i>	
10	<i>The Protocol Layer forms the Wait Message that is passed to the PHY Layer.</i>	
11	<i>PHY Layer appends CRC and sends the Wait Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the Wait Message and compares the CRC it calculated with the one sent to verify the Message.</i>
12		<i>PHY Layer forwards the Wait Message to the Protocol Layer.</i>
13		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</i> <i>Protocol Layer informs the Policy Engine that an Wait Message has been received. The Policy Engine informs the DPM that the Request has been rejected.</i>
14		<i>The Protocol Layer generates a GoodCRC Message and passes it to its PHY Layer.</i>

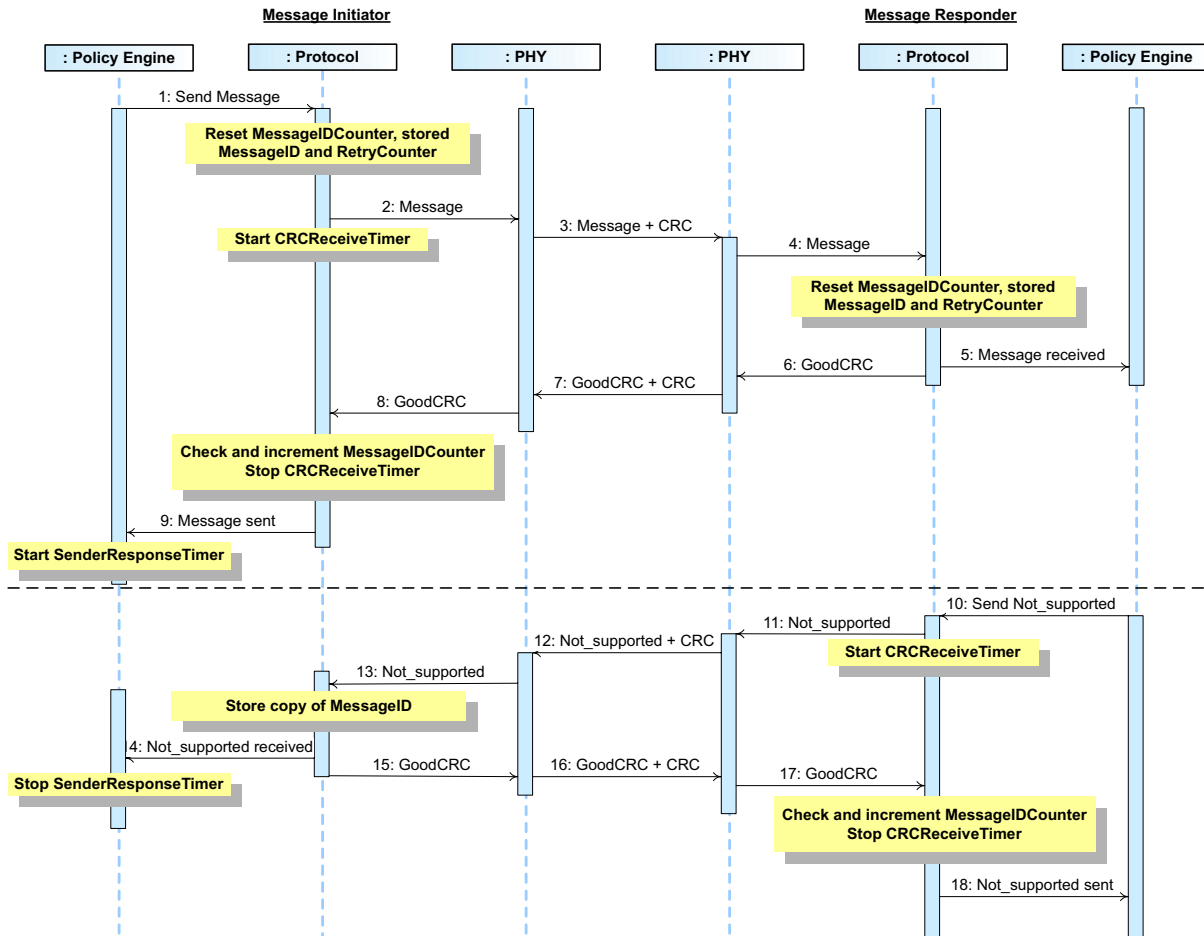
Table 8.50 Steps for SPR Sink Makes Request (Wait)

Step	Source	Sink
15	<i>PHY Layer</i> receives the GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
16	<i>PHY Layer</i> forwards the GoodCRC Message to the <i>Protocol Layer</i> . The <i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops the CRCReceiveTimer .	
17	The <i>Protocol Layer</i> informs the <i>Policy Engine</i> that a Wait Message was successfully sent.	

8.3.2.3 Unsupported Message

This is an example of the response to an *Unsupported Message*. [Figure 8.24, "Unsupported message"](#) shows the *Messages* as they flow across the bus and within the devices.

Figure 8.24 Unsupported message



[Table 8.51, "Steps for an Unsupported Message"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.24, "Unsupported message"](#) above.

Table 8.51 Steps for an Unsupported Message

Step	Message Initiator	Message Responder
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a <i>Message</i> .	
2	<i>Protocol Layer</i> resets <i>MessageIDCounter</i> , stored <i>MessageID</i> and <i>RetryCounter</i> . <i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Message</i> . Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Message</i> to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Message</i> information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Message</i> was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Not_Supported</i> <i>Message</i> .
11		<i>Protocol Layer</i> creates the <i>Not_Supported</i> <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Not_Supported</i> <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Not_Supported</i> <i>Message</i> . Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming <i>Message</i> .	
14	The <i>Protocol Layer</i> forwards the received <i>Not_Supported</i> <i>Message</i> information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> <i>Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC</i> <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

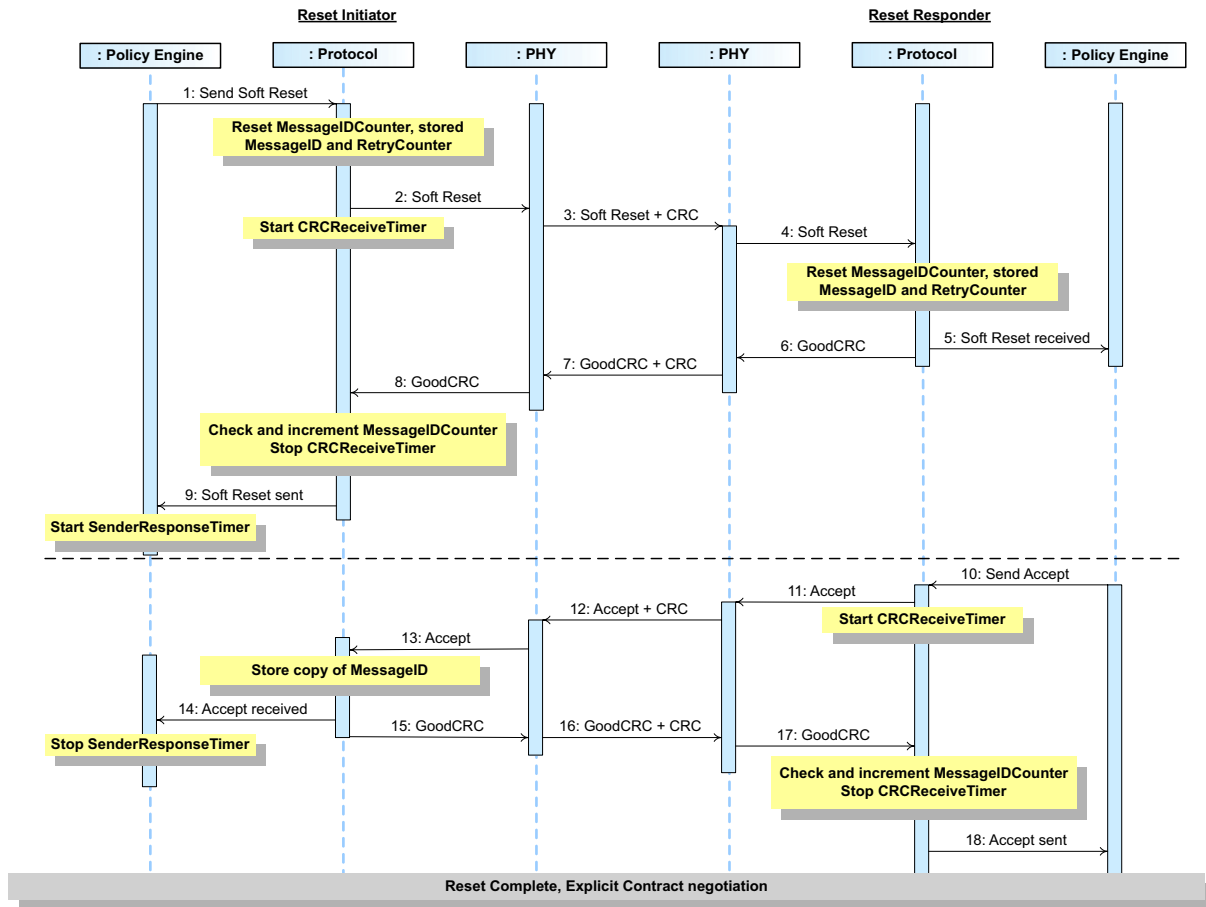
Table 8.51 Steps for an Unsupported Message

Step	Message Initiator	Message Responder
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Not_Supported Message</i> was successfully sent.

8.3.2.4 Soft Reset

This is an example of a *Soft Reset* operation. [Figure 8.25, "Soft Reset"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Soft Reset*.

Figure 8.25 Soft Reset



[Table 8.52, "Steps for a Soft Reset"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.25, "Soft Reset"](#) above.

Table 8.52 Steps for a Soft Reset

Step	Reset Initiator	Reset Responder
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Soft_Reset Message to request a Soft Reset.	
2	<i>Protocol Layer</i> resets MessageIDCounter , stored MessageID and RetryCounter . <i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Soft_Reset Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Soft_Reset Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Soft_Reset Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> does not check the MessageID in the incoming Message and resets MessageIDCounter , stored MessageID and RetryCounter . The <i>Protocol Layer</i> forwards the received Soft_Reset Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Soft_Reset Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .

Table 8.52 Steps for a Soft Reset

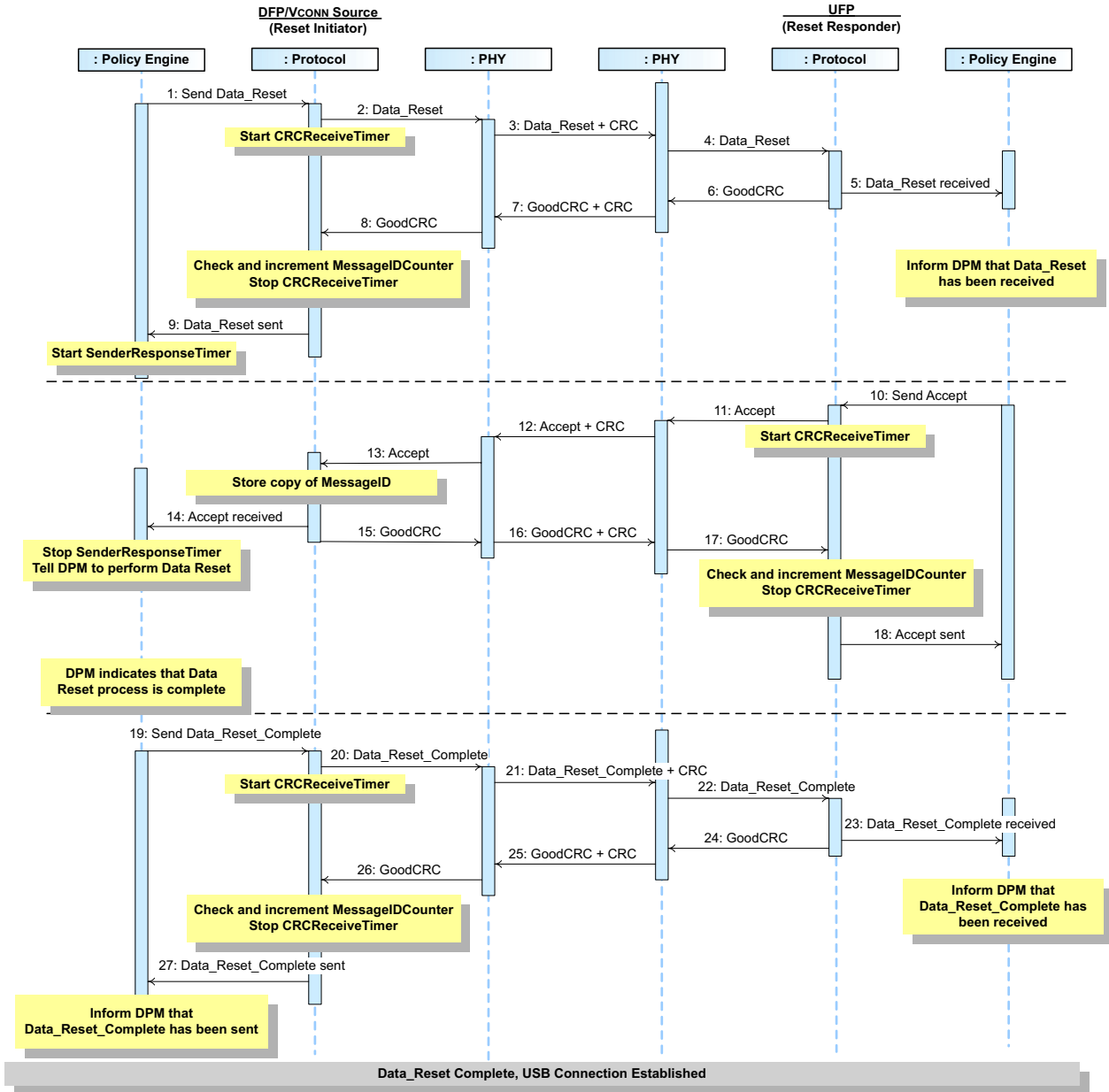
Step	Reset Initiator	Reset Responder
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept Message</i> was successfully sent.
The reset is complete and protocol communication can restart. <i>Port Partners</i> perform an <i>Explicit Contract Negotiation</i> to re-synchronize their state machines.		

8.3.2.5 Data Reset

8.3.2.5.1 DFP Initiated Data Reset where the DFP is the VCONN Source

This is an example of a *Data Reset* operation where the *DFP* is also the *VCONN Source* and initiates a *Data Reset*. [Figure 8.26, "DFP Initiated Data Reset where the DFP is the VCONN Source"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Data Reset*.

Figure 8.26 DFP Initiated Data Reset where the DFP is the VCONN Source



[Table 8.53, "Steps for a DFP Initiated Data Reset where the DFP is the VCONN Source"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.26, "DFP Initiated Data Reset where the DFP is the VCONN Source"](#) above.

Table 8.53 Steps for a DFP Initiated Data Reset where the DFP is the VCONN Source

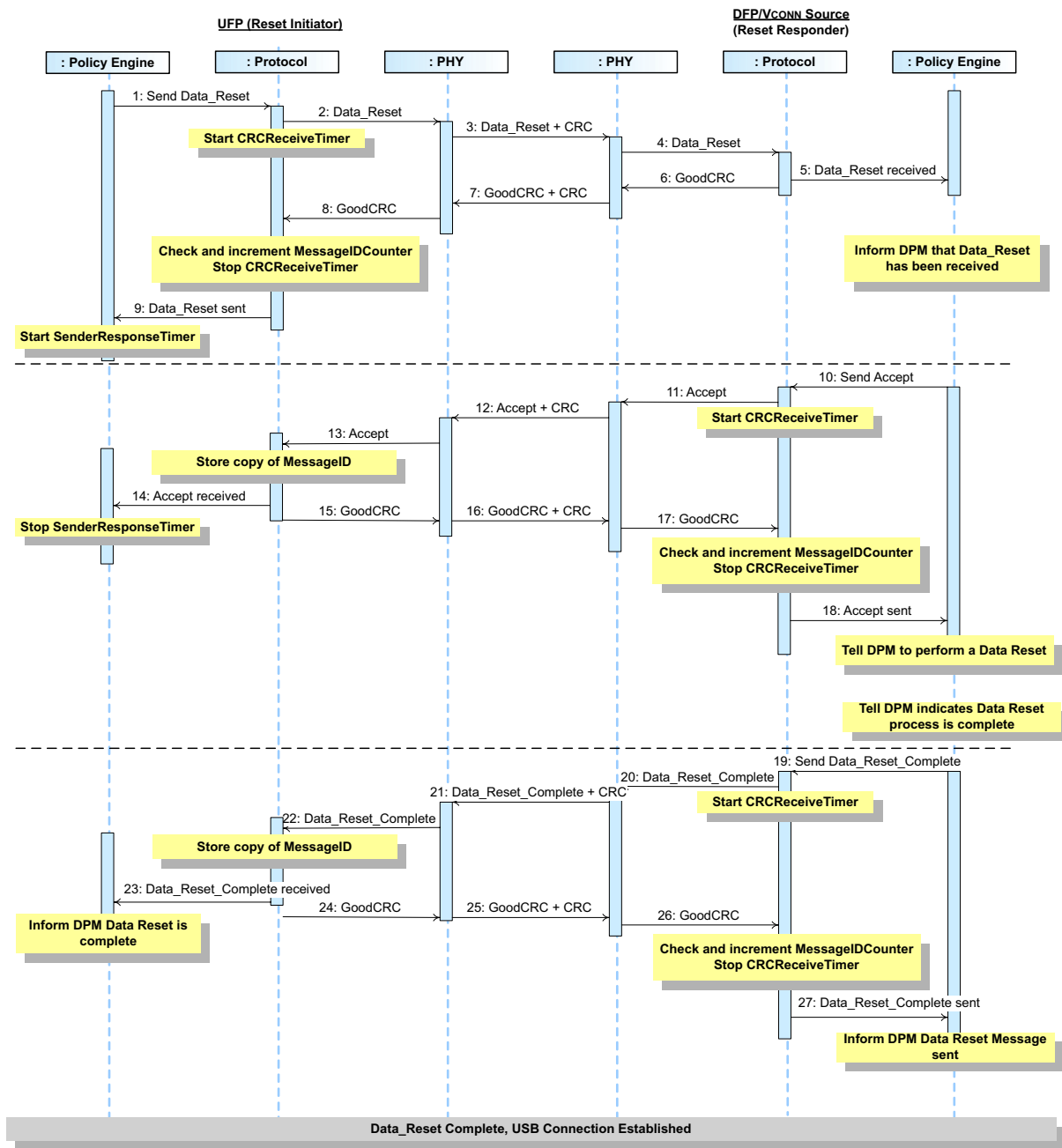
Step	DFP/VCONN Source (Reset Initiator)	UFP (Reset Responder)
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Data_Reset Message to request a <i>Data Reset</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Data_Reset Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Data_Reset Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Data_Reset Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Data_Reset Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset Message has been received.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the SenderResponseTimer and tells the <i>DPM</i> to perform a <i>Data Reset</i> . The <i>DPM</i> proceeds to cycle <i>VCONN</i> and then reset the data connection.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.53 Steps for a DFP Initiated Data Reset where the DFP is the VCONN Source

Step	DFP/VCONN Source (Reset Initiator)	UFP (Reset Responder)
16	PHY Layer appends a CRC and sends the GoodCRC Message.	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
18		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Accept Message was successfully sent.
19	The DPM indicates that the Data Reset process is complete. The Policy Engine directs the Protocol Layer to generate a Data_Reset_Complete Message.	
20	Protocol Layer creates the Message and passes to PHY Layer.	
21	PHY Layer appends CRC and sends the Data_Reset_Complete Message. Starts CRCReceiveTimer .	PHY Layer receives the Data_Reset_Complete Message and compares the CRC it calculated with the one sent to verify the Message.
22		PHY Layer removes the CRC and forwards the Data_Reset_Complete Message to the Protocol Layer.
23		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Data_Reset_Complete Message information to the Policy Engine that consumes it. The Policy Engine informs the DPM that a Data_Reset_Complete Message has been received.
24		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
25	PHY Layer receives the GoodCRC and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
26	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
27	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Data_Reset_Complete Message was successfully sent. The Policy Engine informs the DPM that the Data_Reset_Complete Message was successfully sent.	
The Data Reset is complete as defined in Section 6.3.14, "Data_Reset Message" Step 5. Port Partners re-establish a USB data connection.		

This is an example of a *Data Reset* operation where the *DFP* receives a *Data_Reset* Message and is the *VCONN Source*. [Figure 8.27, "DFP Receives Data Reset where the DFP is the VCONN Source"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Data Reset*.

Figure 8.27 DFP Receives Data Reset where the DFP is the VCONN Source



[Table 8.54, "Steps for a DFP Receiving a Data Reset where the DFP is the VCONNSource"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.27, "DFP Receives Data Reset where the DFP is the VCONN Source"](#) above.

Table 8.54 Steps for a DFP Receiving a Data Reset where the DFP is the VCONNSource

Step	UFP (Reset Initiator)	DFP/VCONN Source (Reset Responder)
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Data_Reset Message to request a <i>Data Reset</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Data_Reset Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Data_Reset Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Data_Reset Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Data_Reset Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset Message has been received.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the SenderResponseTimer . The <i>DPM</i> proceeds to cycle <i>VCONN</i> and then reset the data connection.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

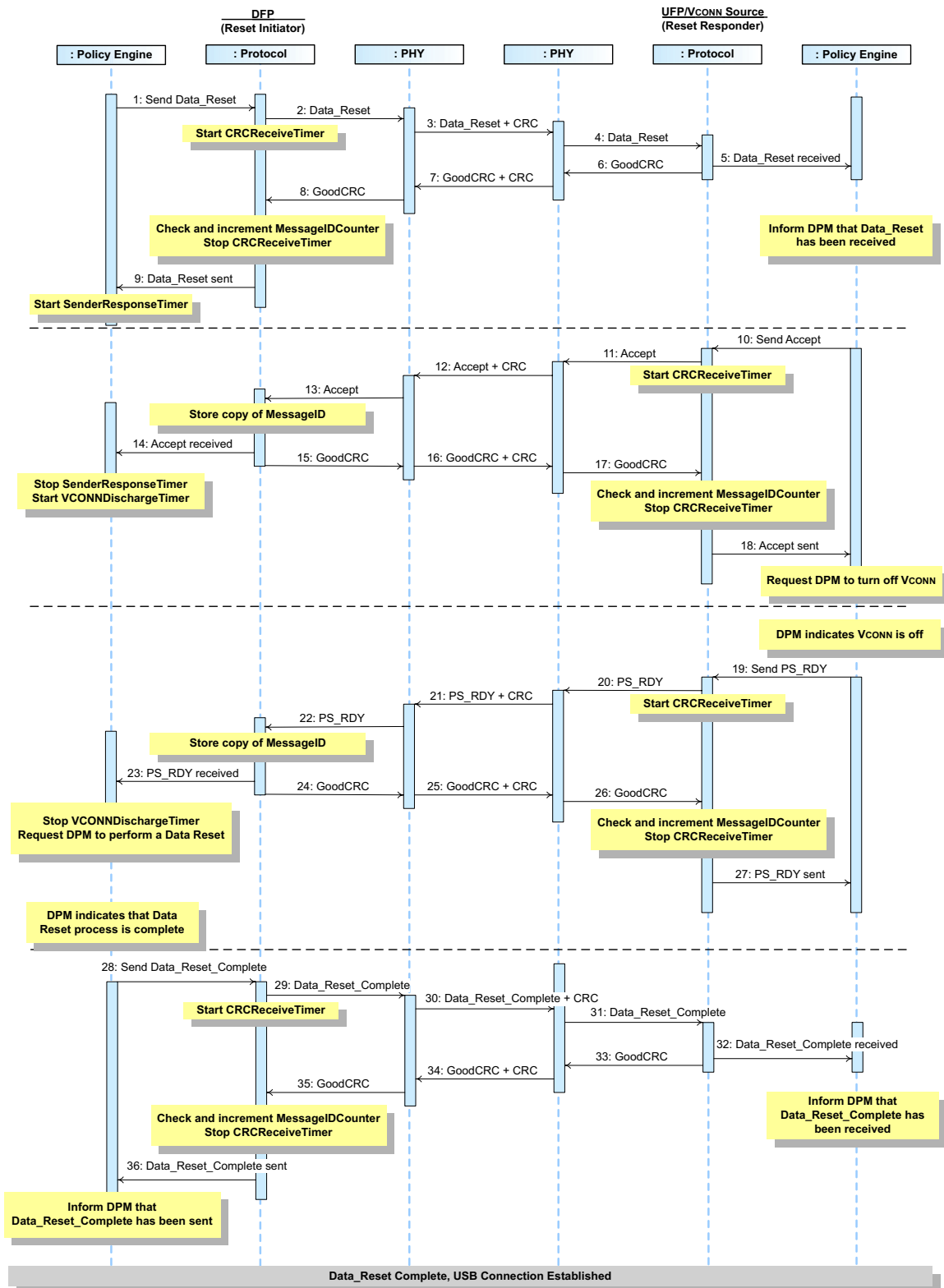
Table 8.54 Steps for a DFP Receiving a Data Reset where the DFP is the VCONN Source

Step	UFP (Reset Initiator)	DFP/VCONN Source (Reset Responder)
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Accept Message was successfully sent. The <i>Policy Engine</i> tells the <i>DPM</i> to perform a <i>Data Reset</i> .
19		The <i>DPM</i> indicates that the <i>Data Reset</i> process is complete. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Data_Reset_Complete Message.
20		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
21	<i>PHY Layer</i> receives the Data_Reset_Complete Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the Data_Reset_Complete Message. Starts CRCReceiveTimer .
22	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Data_Reset_Complete Message to the <i>Protocol Layer</i> .	
23	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Data_Reset_Complete Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset_Complete Message has been received.	
24	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
25	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the Message.
26		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
27		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset_Complete Message was successfully sent. The <i>Policy Engine</i> informs the <i>DPM</i> that the Data_Reset_Complete Message was successfully sent.
The reset is complete as defined in Section 6.3.14, "Data_Reset Message" Step 5. Port Partners re-establish a USB data connection.		

DFP Initiated Data Reset where the UFP is the VCONN Source

Figure 8.28, "DFP Initiated Data Reset where the UFP is the VCONN Source" shows the Messages as they flow across the bus and within the devices to accomplish the Data Reset.

Figure 8.28 DFP Initiated Data Reset where the UFP is the VCONN Source



[Table 8.55, "Steps for a DFP Initiated Data Reset where the UFP is the VCONN Source"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.28, "DFP Initiated Data Reset where the UFP is the VCONN Source"](#) above.

Table 8.55 Steps for a DFP Initiated Data Reset where the UFP is the VCONN Source

Step	DFP (Reset Initiator)	UFP/VCONN Source (Reset Responder)
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Data_Reset Message to request a <i>Soft Reset</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Data_Reset Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Data_Reset Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Data_Reset Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Data_Reset Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset Message has been received.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the SenderResponseTimer and starts the VCONNDischargeTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.55 Steps for a DFP Initiated Data Reset where the UFP is the VCONN Source

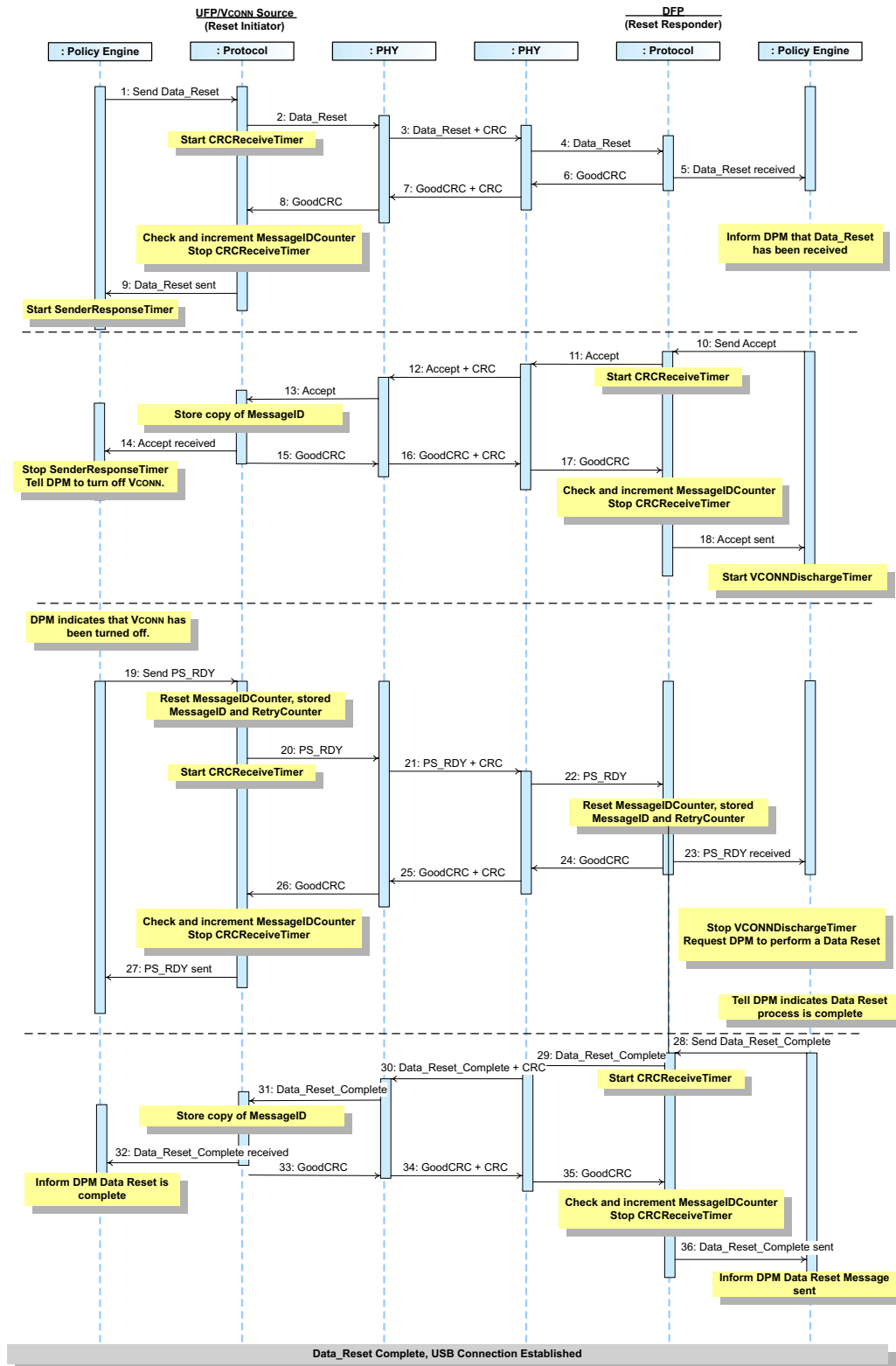
Step	DFP (Reset Initiator)	UFP/VCONN Source (Reset Responder)
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Accept Message was successfully sent. The <i>Policy Engine</i> requests the <i>DPM</i> to turn off <i>VCONN</i> .
19		When the <i>DPM</i> indicates <i>VCONN</i> has been turned off the <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an PS_RDY Message.
20		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
21	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Accept Message. Starts CRCReceiveTimer .
22	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
23	The <i>Protocol Layer</i> forwards the received PS_RDY Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the VCONNDischargeTimer and tells the <i>DPM</i> to perform a Data Reset. The <i>DPM</i> proceeds to turn on <i>VCONN</i> and then reset the data connection.	
24	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
25	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
26		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
27		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent.
28	The <i>DPM</i> indicates that the Data Reset process is complete. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Data_Reset_Complete Message.	
29	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
30	<i>PHY Layer</i> appends <i>CRC</i> and sends the Data_Reset_Complete Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Data_Reset_Complete Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
31		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Data_Reset_Complete Message to the <i>Protocol Layer</i> .

Table 8.55 Steps for a DFP Initiated Data Reset where the UFP is the VCONN Source

Step	DFP (Reset Initiator)	UFP/VCONN Source (Reset Responder)
32		<p><i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received Data_Reset_Complete <i>Message</i> information to the <i>Policy Engine</i> that consumes it.</p> <p>The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset_Complete <i>Message</i> has been received.</p>
33		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
34	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .
35	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
36	<p><i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset_Complete <i>Message</i> was successfully sent.</p> <p>The <i>Policy Engine</i> informs the <i>DPM</i> that the Data_Reset_Complete <i>Message</i> was successfully sent.</p>	
The reset is complete as defined in Section 6.3.14, "Data_Reset Message" Step 5. Port Partners re-establish a USB data connection.		

This is an example of a *Data Reset* operation where the *DFP* receives a *Data_Reset* Message and the *UFP* is the *VCONN* Source. [Figure 8.29, "DFP Receives a Data Reset where the UFP is the VCONN Source"](#) shows the Messages as they flow across the bus and within the devices to accomplish the *Data Reset*.

Figure 8.29 DFP Receives a Data Reset where the UFP is the VCONN Source



[Table 8.56, "Steps for a DFP Receiving a Data Reset where the UFP is the VCONN Source"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.29, "DFP Receives a Data Reset where the UFP is the VCONN Source"](#) above.

Table 8.56 Steps for a DFP Receiving a Data Reset where the UFP is the VCONN Source

Step	UFP/VCONN Source (Reset Initiator)	DFP (Reset Responder)
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a Data_Reset Message to request a <i>Soft Reset</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Data_Reset Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Data_Reset Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Data_Reset Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Data_Reset Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset Message has been received.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the SenderResponseTimer and tells the <i>DPM</i> to turn off <i>VCONN</i> .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.56 Steps for a DFP Receiving a Data Reset where the UFP is the VCONN Source

Step	UFP/VCONN Source (Reset Initiator)	DFP (Reset Responder)
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept</i> Message was successfully sent. The <i>Policy Engine</i> starts the <i>VCONNDischargeTimer</i> .
19	When the <i>DPM</i> indicates that <i>VCONN</i> has been turned off the <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a <i>PS_RDY</i> Message to request a <i>Soft Reset</i> .	
20	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
21	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>PS_RDY</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>PS_RDY</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
22		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>PS_RDY</i> Message to the <i>Protocol Layer</i> .
23		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>PS_RDY</i> Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the <i>VCONNDischargeTimer</i> and requests the <i>DPM</i> perform a <i>Data Reset</i> . The <i>DPM</i> proceeds to turn on <i>VCONN</i> and then reset the data connection.
24		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
25	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
26	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
27	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>PS_RDY</i> Message was successfully sent.	
28		The <i>DPM</i> indicates that the <i>Data Reset</i> process is complete. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a <i>Data_Reset_Complete</i> Message.
29		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
30	<i>PHY Layer</i> receives the <i>Data_Reset_Complete</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Data_Reset_Complete</i> Message. Starts <i>CRCReceiveTimer</i> .
31	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Data_Reset_Complete</i> Message to the <i>Protocol Layer</i> .	

Table 8.56 Steps for a DFP Receiving a Data Reset where the UFP is the VCONN Source

Step	UFP/VCONN Source (Reset Initiator)	DFP (Reset Responder)
32	<p><i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received Data_Reset_Complete <i>Message</i> information to the <i>Policy Engine</i> that consumes it.</p> <p>The <i>Policy Engine</i> informs the <i>DPM</i> that a Data_Reset_Complete <i>Message</i> has been received.</p>	
33	<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .	
34	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the <i>Message</i> .
35		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .
36		<p><i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Data_Reset_Complete <i>Message</i> was successfully sent.</p> <p>The <i>Policy Engine</i> informs the <i>DPM</i> that the Data_Reset_Complete <i>Message</i> was successfully sent.</p>
The reset is complete as defined in Section 6.3.14, "Data_Reset Message" Step 5. Port Partners re-establish a USB data connection.		

8.3.2.6 Hard Reset

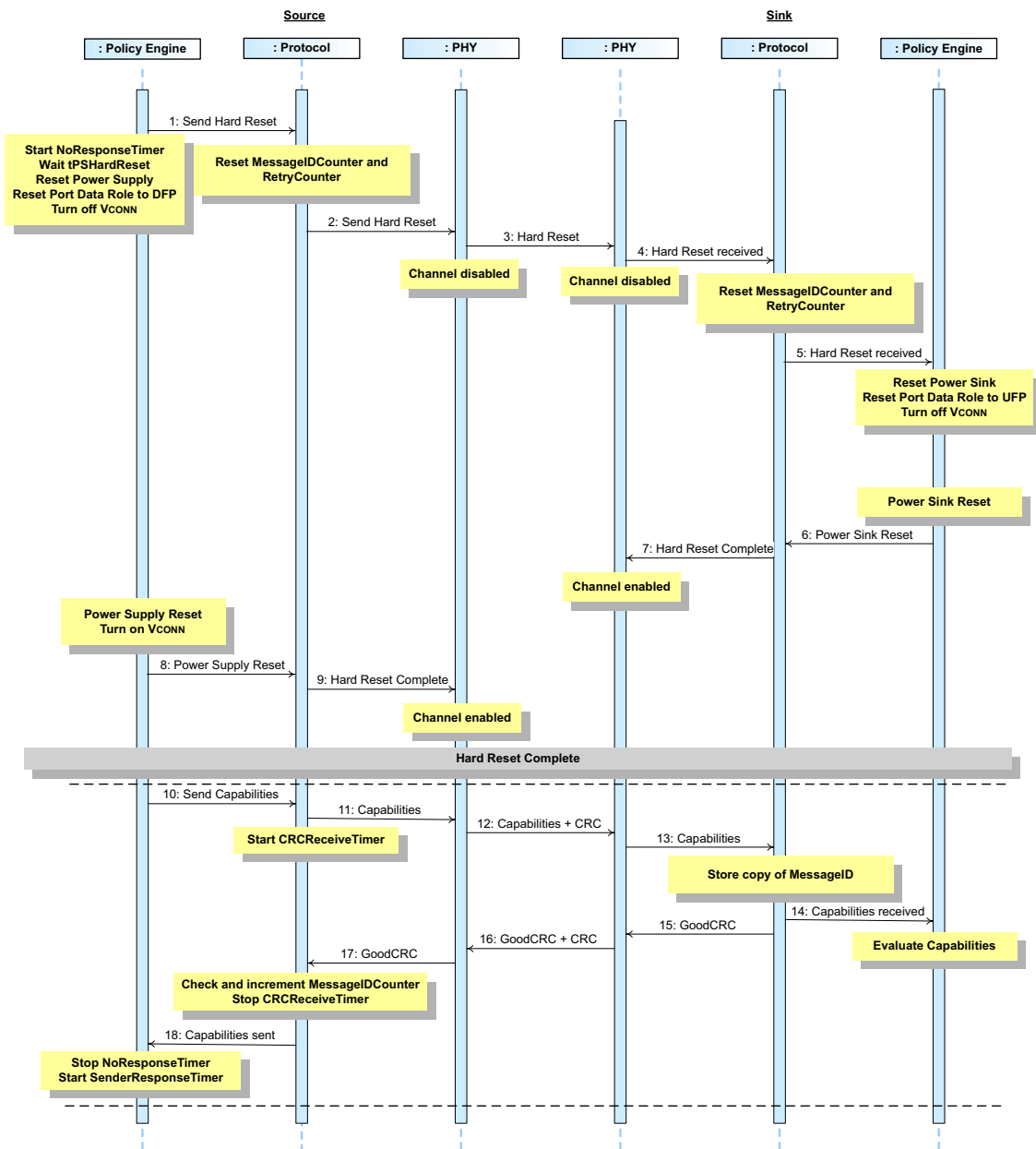
The following sections describe the steps required for a USB Power Delivery *Hard Reset*. The *Hard Reset* returns the operation of the USB Power Delivery to default *Power Role/Data Role* and operating voltage/current. During the *Hard Reset* USB Power Delivery *PHY Layer* communications **shall** be disabled preventing communication between the *Port Partner*.

Note: *Hard Reset*, in this case, is applied to the USB Power Delivery capability of an individual *Port* on which the *Hard Reset* is requested. A side effect of the *Hard Reset* is that it might reset other functions on the *Port* such as USB.

8.3.2.6.1 Source Initiated Hard Reset

This is an example of a *Hard Reset* operation when initiated by a *Source*. [Figure 8.30, "Source initiated Hard Reset"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Hard Reset*.

Figure 8.30 Source initiated Hard Reset



[Table 8.57, "Steps for Source initiated Hard Reset"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.30, "Source initiated Hard Reset"](#) above.

Table 8.57 Steps for Source initiated Hard Reset

Step	Source	Sink
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate Hard Reset Signaling . The <i>Policy Engine</i> starts the NoResponseTimer and requests the <i>DPM</i> to reset the power supply to <i>USB Default Operation</i> . The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Port Data Role to <i>DFP</i> and to turn off <i>VCONN</i> if this is on.	
2	<i>Protocol Layer</i> resets MessageIDCounter and RetryCounter . <i>Protocol Layer</i> requests the <i>PHY Layer</i> send Hard Reset Signaling .	
3	<i>PHY Layer</i> sends Hard Reset Signaling and then disables the <i>PHY Layer</i> communications channel for transmission and reception.	<i>PHY Layer</i> receives the Hard Reset Signaling and disables the <i>PHY Layer</i> communications channel for transmission and reception.
4		<i>PHY Layer</i> informs the <i>Protocol Layer</i> of the Hard Reset . <i>Protocol Layer</i> resets MessageIDCounter and RetryCounter .
5		The <i>Protocol Layer</i> informs the <i>Policy Engine</i> of the Hard Reset . The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Power Sink to <i>USB Default Operation</i> . The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Port Data Role to <i>UFP</i> and to turn off <i>VCONN</i> if this is on.
6		The Power Sink returns to <i>USB Default Operation</i> . The <i>Policy Engine</i> informs the <i>Protocol Layer</i> that the Power Sink has been reset.
7		The <i>Protocol Layer</i> informs the <i>PHY Layer</i> that the Hard Reset is complete. The <i>PHY Layer</i> enables the <i>PHY Layer</i> communications channel for transmission and reception.
8	The power supply is reset to <i>USB Default Operation</i> , and <i>VCONN</i> is turned on. The <i>Policy Engine</i> informs the <i>Protocol Layer</i> that the power supply has been reset.	
9	The <i>Protocol Layer</i> informs the <i>PHY Layer</i> that the Hard Reset is complete. The <i>PHY Layer</i> enables the <i>PHY Layer</i> communications channel for transmission and reception.	
The reset is complete and protocol communication can restart.		
10	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Source_Capabilities Message that represents the power supply's present capabilities.	
11	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
12	<i>PHY Layer</i> appends <i>CRC</i> and sends the Source_Capabilities Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Source_Capabilities Message and checks the <i>CRC</i> to verify the Message.

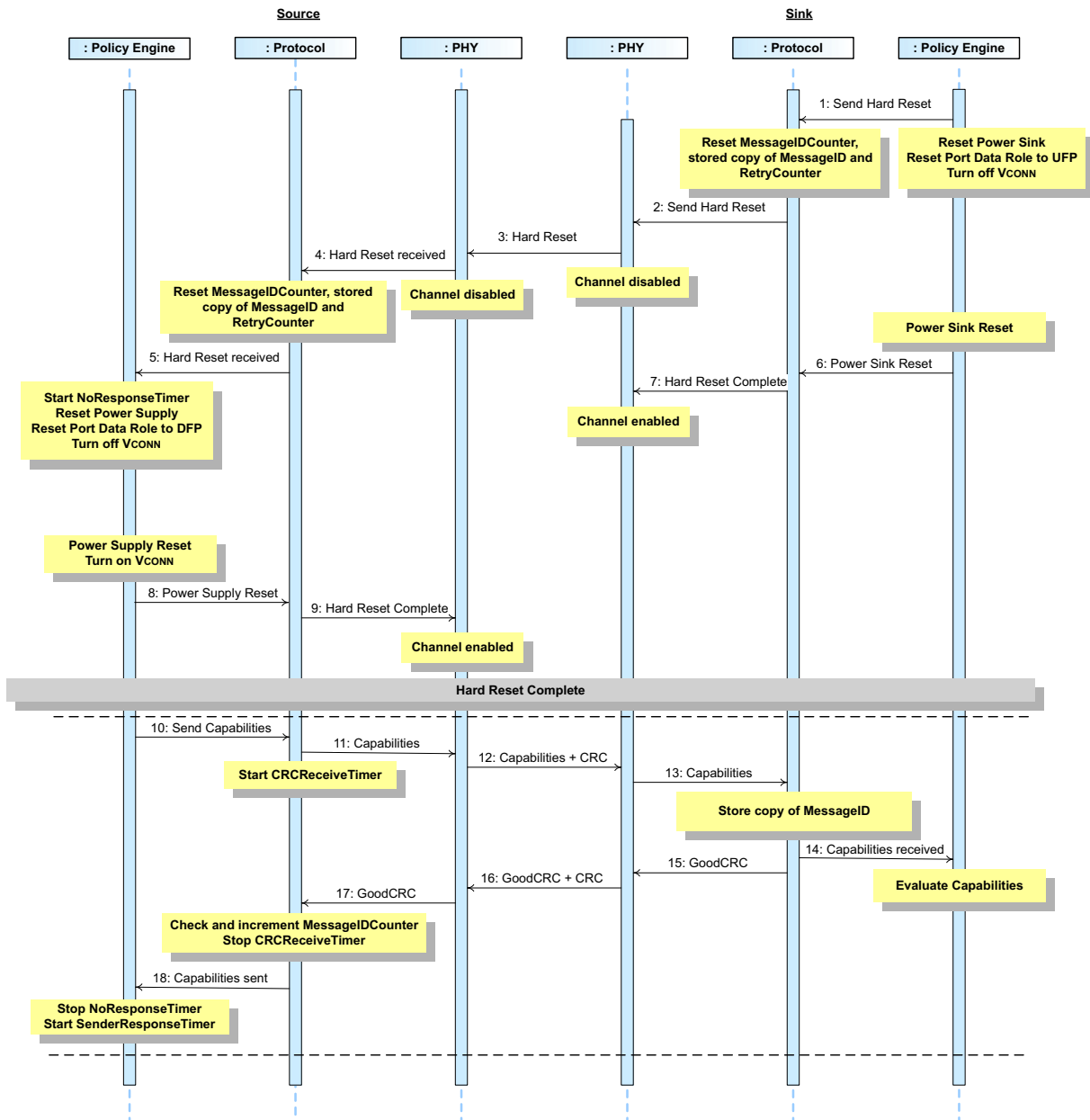
Table 8.57 Steps for Source initiated Hard Reset

Step	Source	Sink
13		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
14		<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming Message. The <i>Protocol Layer</i> forwards the received <i>Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
15		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
16	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
17	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> stops the <i>NoResponseTimer</i> and starts the <i>SenderResponseTimer</i> .	
USB Power Delivery communication is re-established.		

8.3.2.6.2 Sink Initiated Hard Reset

This is an example of a *Hard Reset* operation when initiated by a *Sink*. [Figure 8.31, "Sink Initiated Hard Reset"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Hard Reset*.

Figure 8.31 Sink Initiated Hard Reset



[Table 8.58, "Steps for Sink initiated Hard Reset"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.31, "Sink Initiated Hard Reset"](#) above.

Table 8.58 Steps for Sink initiated Hard Reset

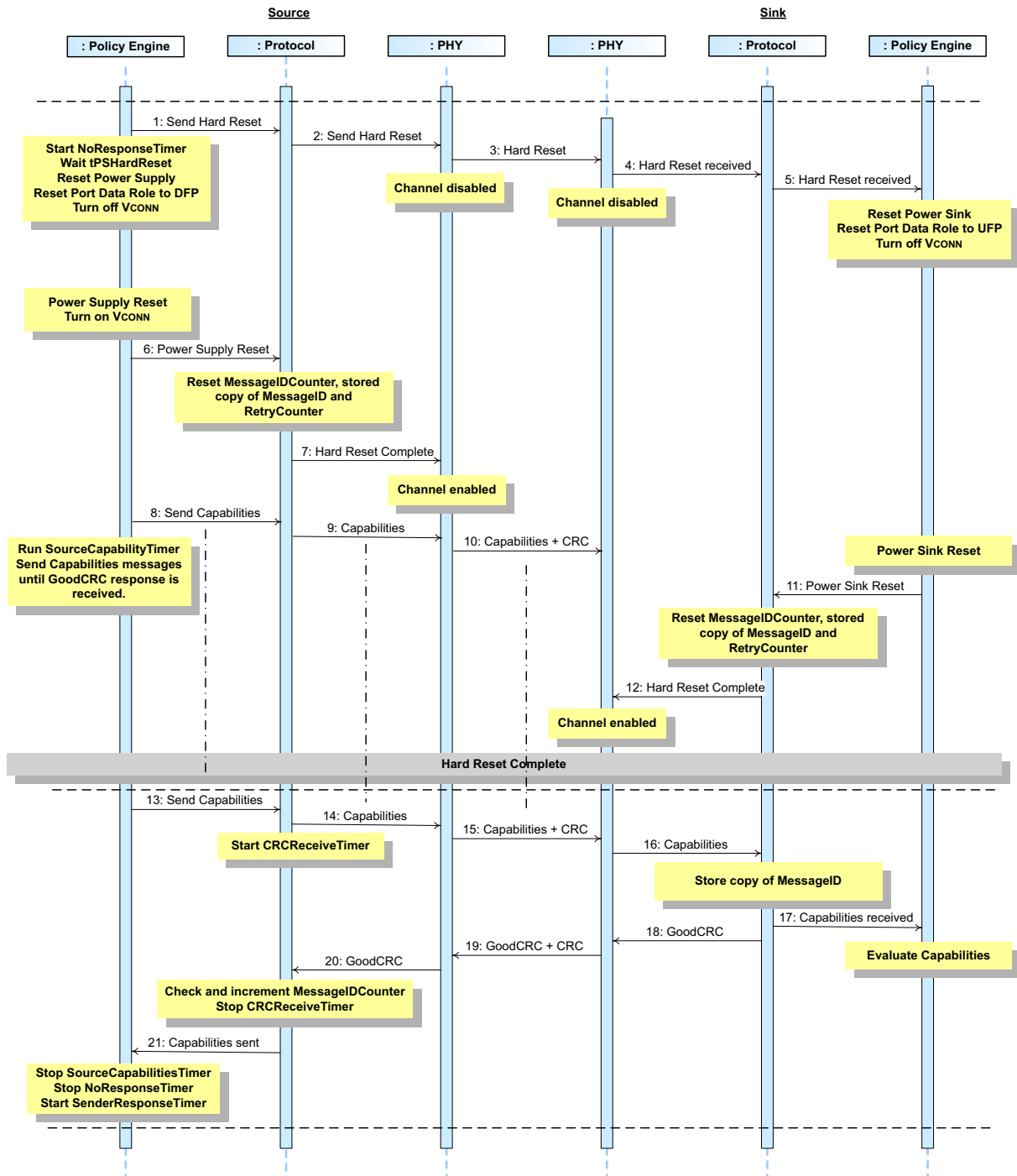
Step	Source	Sink
1		<p>The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate Hard Reset Signaling.</p> <p>The <i>Policy Engine</i> requests the <i>DPM</i> to reset the power supply to <i>USB Default Operation</i>. The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Port Data Role to <i>UFP</i> and to turn off <i>VCONN</i> if this is on.</p>
2		<p><i>Protocol Layer</i> resets MessageIDCounter, stored copy of MessageID and RetryCounter.</p> <p><i>Protocol Layer</i> requests the <i>PHY Layer</i> send Hard Reset Signaling.</p>
3	<i>PHY Layer</i> receives the Hard Reset Signaling and disables the <i>PHY Layer</i> communications channel for transmission and reception.	<i>PHY Layer</i> sends the Hard Reset Signaling and then disables the <i>PHY Layer</i> communications channel for transmission and reception.
4	<p><i>PHY Layer</i> informs the <i>Protocol Layer</i> of the Hard Reset.</p> <p><i>Protocol Layer</i> resets MessageIDCounter, stored copy of MessageID and RetryCounter.</p>	
5	<p>The <i>Protocol Layer</i> Informs the <i>Policy Engine</i> of the Hard Reset.</p> <p>The <i>Policy Engine</i> starts the NoResponseTimer and requests the <i>DPM</i> to reset the Power Sink to <i>USB Default Operation</i>. The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Port Data Role to <i>DFP</i> and to turn off <i>VCONN</i> if this is on.</p>	
6		<p>The Power Sink returns to <i>USB Default Operation</i>.</p> <p>The <i>Policy Engine</i> informs the <i>Protocol Layer</i> that the Power Sink has been reset.</p>
7		<p>The <i>Protocol Layer</i> informs the <i>PHY Layer</i> that the Hard Reset is complete.</p> <p>The <i>PHY Layer</i> enables the <i>PHY Layer</i> communications channel for transmission and reception.</p>
8	<p>The power supply is reset to <i>USB Default Operation</i> and <i>VCONN</i> is turned on.</p> <p>The <i>Policy Engine</i> informs the <i>Protocol Layer</i> that the power supply has been reset.</p>	
9	The <i>Protocol Layer</i> informs the <i>PHY Layer</i> that the Hard Reset is complete. The <i>PHY Layer</i> enables the <i>PHY Layer</i> communications channel for transmission and reception.	
The reset is complete and protocol communication can restart.		
10	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Source_Capabilities Message that represents the power supply's present capabilities.	
11	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
12	<i>PHY Layer</i> appends <i>CRC</i> and sends the Source_Capabilities Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Source_Capabilities Message and checks the <i>CRC</i> to verify the Message.

Table 8.58 Steps for Sink initiated Hard Reset

Step	Source	Sink
13		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Source_Capabilities</i> Message to the <i>Protocol Layer</i> .
14		<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming Message. The <i>Protocol Layer</i> forwards the received <i>Source_Capabilities</i> Message information to the <i>Policy Engine</i> that consumes it.
15		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
16	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
17	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities</i> Message was successfully sent. <i>Policy Engine</i> stops the <i>NoResponseTimer</i> and starts the <i>SenderResponseTimer</i> .	
USB Power Delivery communication is re-established.		

This is an example of a *Hard Reset* operation when initiated by a *Source*. In this example the *Sink* is slow responding to the reset causing the *Source* to send multiple *Source_Capabilities* Messages before it receives a *GoodCRC* Message response. [Figure 8.32, "Source initiated reset - Sink long reset"](#) shows the Messages as they flow across the bus and within the devices to accomplish the *Hard Reset*.

Figure 8.32 Source initiated reset - Sink long reset



[Table 8.59, "Steps for Source initiated Hard Reset - Sink long reset"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.32, "Source initiated reset - Sink long reset"](#) above.

Table 8.59 Steps for Source initiated Hard Reset - Sink long reset

Step	Source	Sink
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate Hard Reset Signaling . The <i>Policy Engine</i> starts the NoResponseTimer and requests the <i>DPM</i> to reset the power supply to <i>USB Default Operation</i> . The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Port Data Role to <i>DFP</i> and to turn off <i>VCONN</i> if this is on.	
2	<i>Protocol Layer</i> resets MessageIDCounter , stored copy of MessageID and RetryCounter . <i>Protocol Layer</i> requests the <i>PHY Layer</i> send Hard Reset Signaling .	
3	<i>PHY Layer</i> sends the Hard Reset Signaling and then disables the <i>PHY Layer</i> communications channel for transmission and reception.	<i>PHY Layer</i> receives the Hard Reset Signaling and disables the <i>PHY Layer</i> communications channel for transmission and reception.
4		<i>PHY Layer</i> informs the <i>Protocol Layer</i> of the Hard Reset . <i>Protocol Layer</i> resets MessageIDCounter , stored copy of MessageID and RetryCounter .
5		The <i>Protocol Layer</i> Informs the <i>Policy Engine</i> of the Hard Reset . The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Power Sink to <i>USB Default Operation</i> . The <i>Policy Engine</i> requests the <i>DPM</i> to reset the Port Data Role to <i>UFP</i> and to turn off <i>VCONN</i> if this is on.
6	The power supply is reset to <i>USB Default Operation</i> and <i>VCONN</i> is turned on. The <i>Policy Engine</i> informs the <i>Protocol Layer</i> that the power supply has been reset.	
7	The <i>Protocol Layer</i> informs the <i>PHY Layer</i> that the Hard Reset is complete. The <i>PHY Layer</i> enables the <i>PHY Layer</i> communications channel for transmission and reception.	
The reset is complete and protocol communication can restart.		
8	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Source_Capabilities Message that represents the power supply's present capabilities. <i>Policy Engine</i> starts the SourceCapabilityTimer . The SourceCapabilityTimer times out one or more times until a GoodCRC Message response is received.	
9	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
10	<i>PHY Layer</i> appends <i>CRC</i> and sends the Source_Capabilities Message. Starts CRCReceiveTimer .	Note: Source_Capabilities Message not received since channel is disabled.
11		The Power Sink returns to <i>USB Default Operation</i> . The <i>Policy Engine</i> informs the <i>Protocol Layer</i> that the Power Sink has been reset.

Table 8.59 Steps for Source initiated Hard Reset - Sink long reset

Step	Source	Sink
12		<p>The <i>Protocol Layer</i> informs the <i>PHY Layer</i> that the <i>Hard Reset</i> is complete.</p> <p>The <i>PHY Layer</i> enables the <i>PHY Layer</i> communications channel for transmission and reception.</p>
The reset is complete and protocol communication can restart.		
13	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>Source_Capabilities</i> <i>Message</i> that represents the power supply's present capabilities. Starts the <i>SourceCapabilityTimer</i> .	
14	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
15	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Source_Capabilities</i> <i>Message</i> . Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Source_Capabilities</i> <i>Message</i> and checks the <i>CRC</i> to verify the <i>Message</i> .
16		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Source_Capabilities</i> <i>Message</i> to the <i>Protocol Layer</i> .
17		<p><i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming <i>Message</i>.</p> <p>The <i>Protocol Layer</i> forwards the received <i>Source_Capabilities</i> <i>Message</i> information to the <i>Policy Engine</i> that consumes it.</p>
18		<i>Protocol Layer</i> generates a <i>GoodCRC</i> <i>Message</i> and passes it <i>PHY Layer</i> .
19	<i>PHY Layer</i> receives the <i>GoodCRC</i> <i>Message</i> and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> <i>Message</i> .
20	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> <i>Message</i> to the <i>Protocol Layer</i> .	
21	<p><i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i>.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities</i> <i>Message</i> was successfully sent.</p> <p><i>Policy Engine</i> stops the <i>SourceCapabilityTimer</i>, stops the <i>NoResponseTimer</i> and starts the <i>SenderResponseTimer</i>.</p>	
USB Power Delivery communication is re-established.		

8.3.2.7 Power Role Swap

8.3.2.7.1 Source Initiated Power Role Swap

8.3.2.7.1.1 Source Initiated Power Role Swap (Accept)

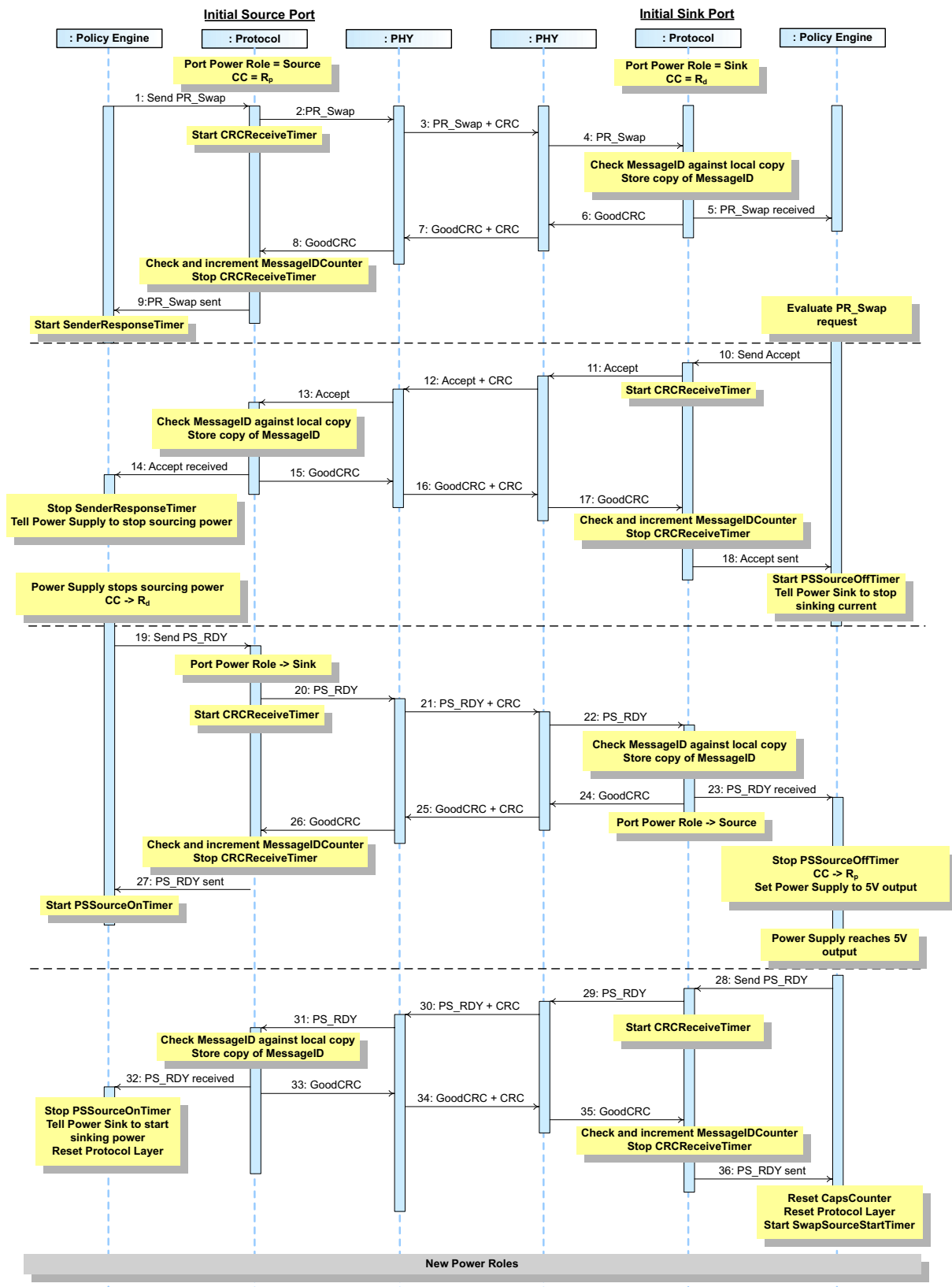
This is an example of a successful *Power Role Swap* operation initiated by a *Port* which initially, at the start of this *Message* sequence, is acting as a *Source* and therefore has R_p pulled up on its *CC* wire. It does not include any subsequent *Power Negotiation* which is required in order to establish an *Explicit Contract* (see [Section 8.3.2.2, "Power Negotiation"](#)).

There are four distinct phases to the *Power Role Swap*:

- A **PR_Swap** Message is sent.
- An **Accept** Message in response to the **PR_Swap** Message.
- The *New Sink* sets its power output to **vSafe0V**, then asserts R_d and sends a **PS_RDY** Message when this process is complete.
- The *New Source* asserts R_p , then sets its power output to **vSafe5V** and sends a **PS_RDY** Message when it is ready to supply power.

[Figure 8.33, "Successful Power Role Swap Sequence Initiated by the Source"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Power Role Swap* sequence.

Figure 8.33 Successful Power Role Swap Sequence Initiated by the Source



[Table 8.60, "Steps for a Successful Source Initiated Power Role Swap Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.33, "Successful Power Role Swap Sequence Initiated by the Source"](#) above.

Table 8.60 Steps for a Successful Source Initiated Power Role Swap Sequence

Step	Initial Source Port	Initially Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a PR_Swap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the PR_Swap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PR_Swap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the PR_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PR_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PR_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the PR_Swap Message sent by the <i>Source</i> and decides that it is able and willing to do the Power Role Swap. It tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Accept Message.	<i>PHY Layer</i> appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> requests its power supply to stop supplying power and stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.60 Steps for a Successful Source Initiated Power Role Swap Sequence

Step	Initial Source Port	Initially Sink Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Accept Message was successfully sent. The Policy Engine starts the PSSourceOffTimer and tells the power supply to stop sinking current.</i>
19	<i>The Policy Engine determines its power supply is no longer supplying VBUS. The Policy Engine requests the DPM to assert the R_d pull down on the CC wire. The Policy Engine then directs the Protocol Layer to generate a PS_RDY Message, with the Port Power Role Message set to Sink, to tell its Port Partner that it can begin to source VBUS.</i>	
20	<i>Protocol Layer sets the Port Power Role Message to Sink, creates the Message and passes to PHY Layer.</i>	
21	<i>PHY Layer appends CRC and sends the PS_RDY Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the PS_RDY Message and checks the CRC to verify the Message.</i>
22		<i>PHY Layer removes the CRC and forwards the PS_RDY Message to the Protocol Layer.</i>
23		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received PS_RDY Message information to the Policy Engine that consumes it. The Policy Engine stops the PSSourceOffTimer, directs the DPM to apply the R_p pull up and then starts switching the power supply to vSafe5V Source operation.</i>
24		<i>Protocol Layer generates a GoodCRC Message and passes it PHY Layer.</i>
25	<i>PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.</i>	<i>PHY Layer appends CRC and sends the GoodCRC Message.</i>
26	<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>	
27	<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the PS_RDY Message was successfully sent. Policy Engine starts PSSourceOnTimer.</i>	
28		<i>Policy Engine, when its power supply is ready to supply power, tells the Protocol Layer to form a PS_RDY Message. The Port Power Role Message is set to Source.</i>
29		<i>Protocol Layer creates the PS_RDY Message and passes to PHY Layer.</i>

Table 8.60 Steps for a Successful Source Initiated Power Role Swap Sequence

Step	Initial Source Port	Initially Sink Port
30	<i>PHY Layer</i> receives the PS_RDY Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the PS_RDY Message. Starts CRCReceiveTimer .
31	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the PS_RDY Message to the <i>Protocol Layer</i> .	
32	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PS_RDY Message information to the <i>Policy Engine</i> that consumes it.	
33	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
34	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message. The <i>Policy Engine</i> stops the PSSourceOnTimer , informs the power supply it can now sink power and resets the <i>Protocol Layer</i> .	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
35		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
36		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent. The <i>Policy Engine</i> resets the CapsCounter , resets the <i>Protocol Layer</i> and starts the SwapSourceStartTimer which must timeout before sending any Source_Capabilities Messages.
The <i>Power Role Swap</i> is complete, the <i>Power Roles</i> have been reversed and the <i>Port Partners</i> are free to <i>Negotiate</i> for more power.		

8.3.2.7.1.2 Source Initiated Power Role Swap (Reject)

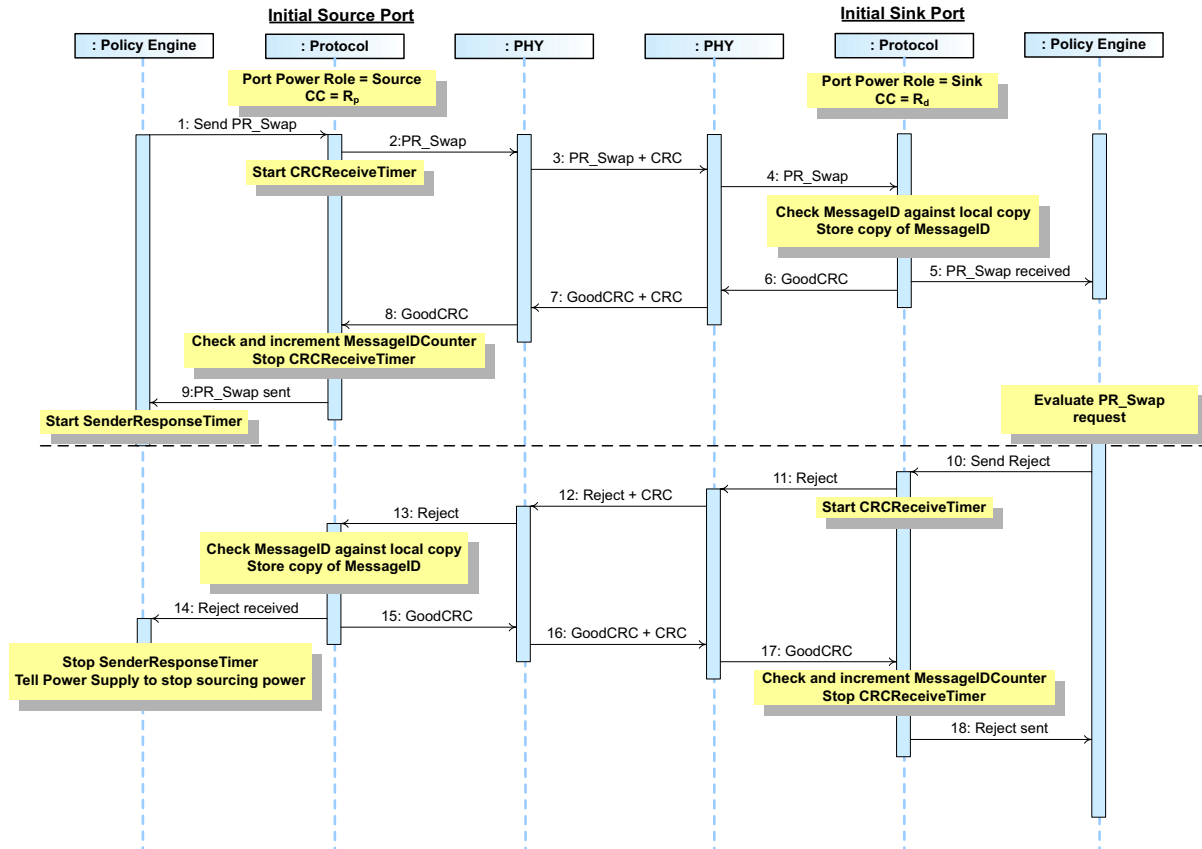
This is an example of a rejected *Power Role Swap* operation initiated by a *Port* which initially, at the start of this *Message* sequence, is acting as a *Source* and therefore has R_p pulled up on its *CC* wire.

There are several phases to the *Power Role Swap*:

- A *PR_Swap* Message is sent.
- An *Reject* Message in response to the *PR_Swap* Message.

Figure 8.34, "Rejected Power Role Swap Sequence Initiated by the Source" shows the Messages as they flow across the bus and within the devices.

Figure 8.34 Rejected Power Role Swap Sequence Initiated by the Source



[Table 8.61, "Steps for a Rejected Source Initiated Power Role Swap Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.34, "Rejected Power Role Swap Sequence Initiated by the Source"](#) above.

Table 8.61 Steps for a Rejected Source Initiated Power Role Swap Sequence

Step	Initial Source Port	Initially Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a PR_Swap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the PR_Swap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PR_Swap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the PR_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PR_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PR_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the PR_Swap Message sent by the <i>Source</i> and decides that it is unable and unwilling to do the <i>Power Role Swap</i> . It tells the <i>Protocol Layer</i> to form a Reject Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Reject Message.	<i>PHY Layer</i> appends a CRC and sends the Reject Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Reject Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.61 Steps for a Rejected Source Initiated Power Role Swap Sequence

Step	Initial Source Port	Initially Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject Message</i> was successfully sent.

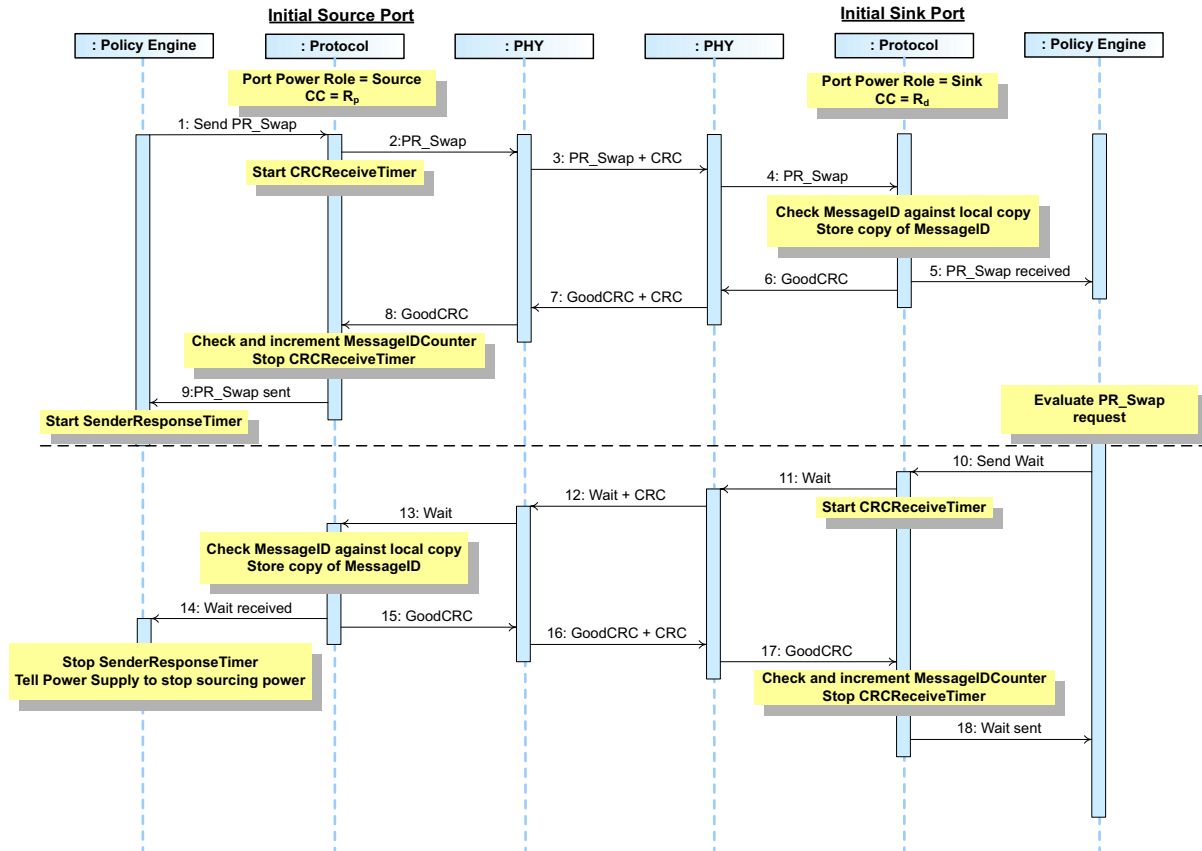
This is an example of a *Power Role Swap* operation, with a wait response, initiated by a *Port* which initially, at the start of this *Message* sequence, is acting as a *Source* and therefore has R_p pulled up on its *CC* wire.

There are several phases to the *Power Role Swap*:

- A *PR_Swap* Message is sent.
- A *Wait* Message in response to the *PR_Swap* Message.

[Figure 8.35, "Power Role Swap Sequence with wait Initiated by the Source"](#) shows the *Messages* as they flow across the bus and within the devices.

Figure 8.35 Power Role Swap Sequence with wait Initiated by the Source



[Table 8.62, "Steps for a Source Initiated Power Role Swap with Wait Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.35, "Power Role Swap Sequence with wait Initiated by the Source"](#) above.

Table 8.62 Steps for a Source Initiated Power Role Swap with Wait Sequence

Step	Initial Source Port	Initially Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a PR_Swap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the PR_Swap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PR_Swap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the PR_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PR_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PR_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the PR_Swap Message sent by the <i>Source</i> and decides that it is able and willing to do the Power Role Swap but not at this time. It tells the <i>Protocol Layer</i> to form a Wait Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Wait Message.	<i>PHY Layer</i> appends a CRC and sends the Wait Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Wait Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.62 Steps for a Source Initiated Power Role Swap with Wait Sequence

Step	Initial Source Port	Initially Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent.

8.3.2.7.2 Sink Initiated Power Role Swap

8.3.2.7.2.1 Sink Initiated Power Role Swap (Accept)

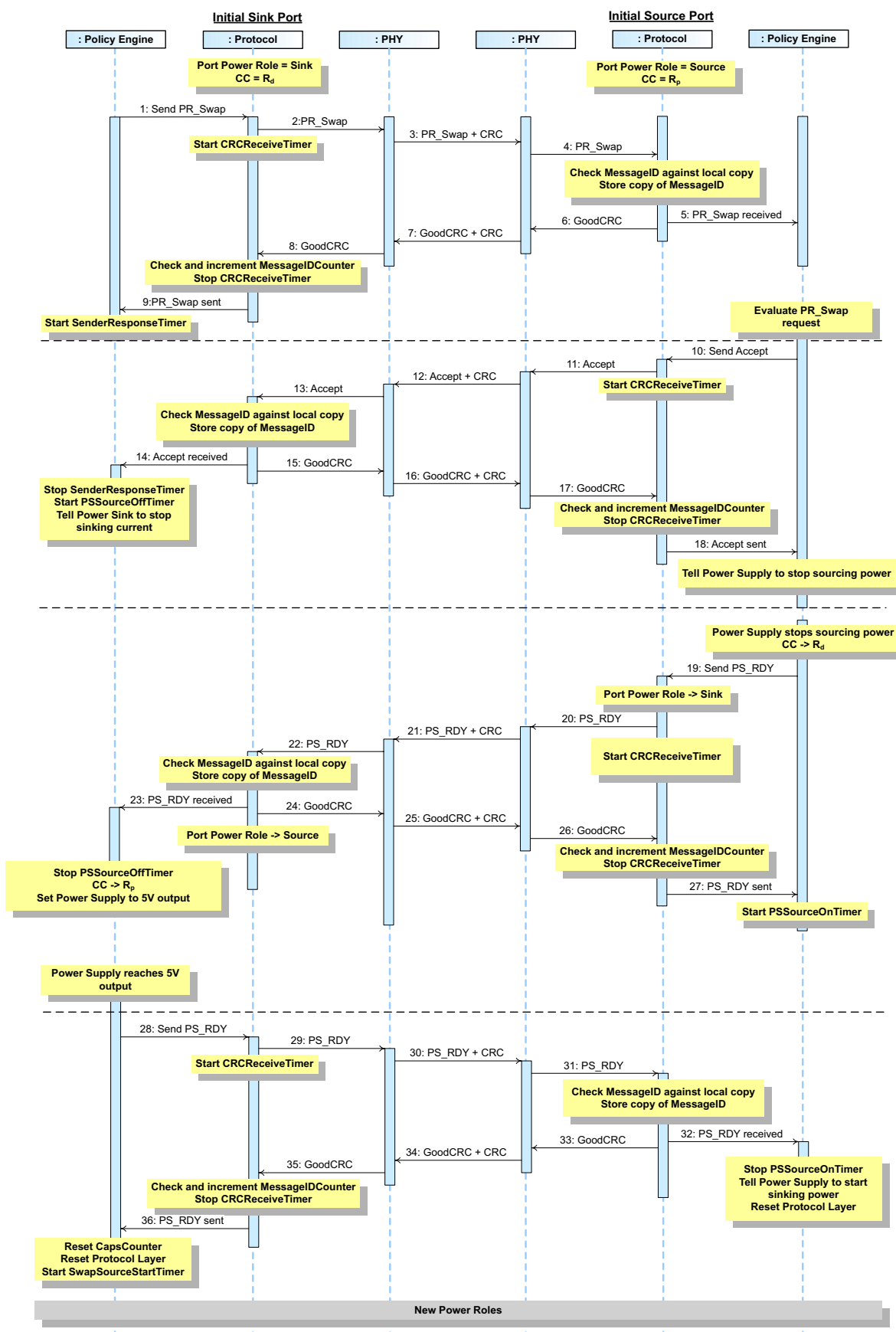
This is an example of a successful *Power Role Swap* operation initiated by a *Port* which initially, at the start of this *Message* sequence, is acting as a *Sink* and therefore has R_d pulled down on its *CC* wire. It does not include any subsequent *Power Negotiation* which is required in order to establish an *Explicit Contract* (see [Section 8.3.2.2, "Power Negotiation"](#)).

There are four distinct phases to the *Power Role Swap*:

- A **PR_Swap** *Message* is sent.
- An **Accept** *Message* in response to the **PR_Swap** *Message*.
- The *New Sink* sets its power output to **vSafe0V**, then asserts R_d and sends a **PS_RDY** *Message* when this process is complete.
- The *New Source* asserts R_p , then sets its power output to **vSafe5V** and sends a **PS_RDY** *Message* when it is ready to supply power.

[Figure 8.36, "Successful Power Role Swap Sequence Initiated by the Sink"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Power Role Swap*.

Universal Serial Bus Power Delivery Specification, Revision 3.2, Version 1.1, 2024-10 Page 549



[Table 8.63, "Steps for a Successful Sink Initiated Power Role Swap Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.36, "Successful Power Role Swap Sequence Initiated by the Sink"](#) above.

Table 8.63 Steps for a Successful Sink Initiated Power Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a PR_Swap Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the PR_Swap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the PR_Swap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the PR_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PR_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PR_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the PR_Swap Message sent by the Sink and decides that it is able and willing to do the Power Role Swap. It tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Accept Message.	<i>PHY Layer</i> appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer , starts the PSSourceOffTimer and tells the power supply to stop sinking current.	

Table 8.63 Steps for a Successful Sink Initiated Power Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Accept Message was successfully sent. The <i>Policy Engine</i> tells the power supply to stop supplying power.
19		The <i>Policy Engine</i> determines its power supply is no longer supplying <i>V_{BUS}</i> . The <i>Policy Engine</i> requests the <i>DPM</i> to assert the <i>R_d</i> pull down on the <i>CC</i> wire. The <i>Policy Engine</i> then directs the <i>Protocol Layer</i> to generate a PS_RDY Message, with the Port Power Role Message set to Sink, to tell its <i>Port Partner</i> that it can begin to source <i>V_{BUS}</i> .
20		<i>Protocol Layer</i> sets the Port Power Role Message to Sink, creates the Message and passes to <i>PHY Layer</i> .
21	<i>PHY Layer</i> receives the PS_RDY Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the PS_RDY Message. Starts CRCReceiveTimer .
22	<i>PHY Layer</i> removes the CRC and forwards the PS_RDY Message to the <i>Protocol Layer</i> .	
23	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PS_RDY Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the PSSourceOffTimer , directs the <i>DPM</i> to apply the <i>R_p</i> pull up and then starts switching the power supply to vSafe5V Source operation.	
24	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
25	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.
26		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
27		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent. <i>Policy Engine</i> starts PSSourceOnTimer .
28	<i>Policy Engine</i> , when its power supply is ready to supply power, tells the <i>Protocol Layer</i> to form a PS_RDY Message. The Port Power Role Message is set to Source.	
29	<i>Protocol Layer</i> creates the PS_RDY Message and passes to <i>PHY Layer</i> .	

Table 8.63 Steps for a Successful Sink Initiated Power Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
30	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>PS_RDY</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>PS_RDY</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
31		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>PS_RDY</i> Message to the <i>Protocol Layer</i> .
32		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>PS_RDY</i> Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the <i>PSSourceOnTimer</i> , informs the power supply that it can start consuming power.
33		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
34	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message. The <i>Policy Engine</i> stops the <i>PSSourceOnTimer</i> , informs the power supply it can now sink power and resets the <i>Protocol Layer</i> .
35	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> to the <i>Protocol Layer</i> .	
36	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>PS_RDY</i> Message was successfully sent. The <i>Policy Engine</i> resets the <i>CapsCounter</i> , resets the <i>Protocol Layer</i> and starts the <i>SwapSourceStartTimer</i> which must timeout before sending any <i>Source_Capabilities</i> Messages.	
The <i>Power Role Swap</i> is complete, the <i>Power Roles</i> have been reversed and the <i>Port Partners</i> are free to <i>Negotiate</i> for more power.		

8.3.2.7.2.2 Sink Initiated Power Role Swap (Reject)

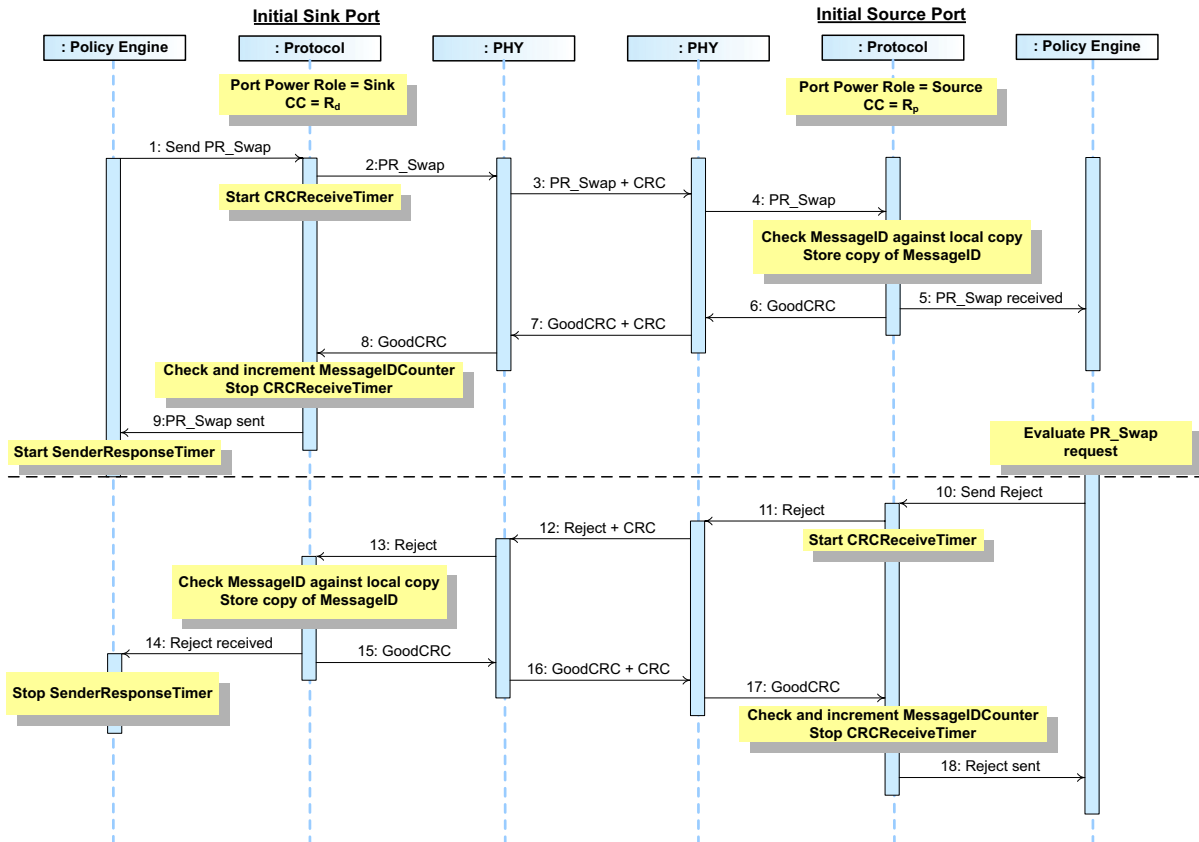
This is an example of a rejected *Power Role Swap* operation initiated by a *Port* which initially, at the start of this *Message* sequence, is acting as a *Sink* and therefore has R_d pulled down on its *CC* wire.

There are several phases to the *Power Role Swap*:

- A *PR_Swap* *Message* is sent.
- A *Reject* *Message* in response to the *PR_Swap* *Message*.

[Figure 8.37, "Rejected Power Role Swap Sequence Initiated by the Sink"](#) shows the *Messages* as they flow across the bus and within the devices.

Figure 8.37 Rejected Power Role Swap Sequence Initiated by the Sink



[Table 8.64, "Steps for a Rejected Sink Initiated Power Role Swap Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.37, "Rejected Power Role Swap Sequence Initiated by the Sink"](#) above.

Table 8.64 Steps for a Rejected Sink Initiated Power Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a PR_Swap Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the PR_Swap Message. Starts CRCReceiveTimer .	PHY Layer receives the PR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the PR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received PR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the PR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the PR_Swap Message sent by the Sink and decides that it is unable and unwilling to do the Power Role Swap. It tells the Protocol Layer to form a Reject Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Reject Message.	PHY Layer appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.64 Steps for a Rejected Sink Initiated Power Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject</i> Message was successfully sent

8.3.2.7.2.3 Sink Initiated Power Role Swap (Wait)

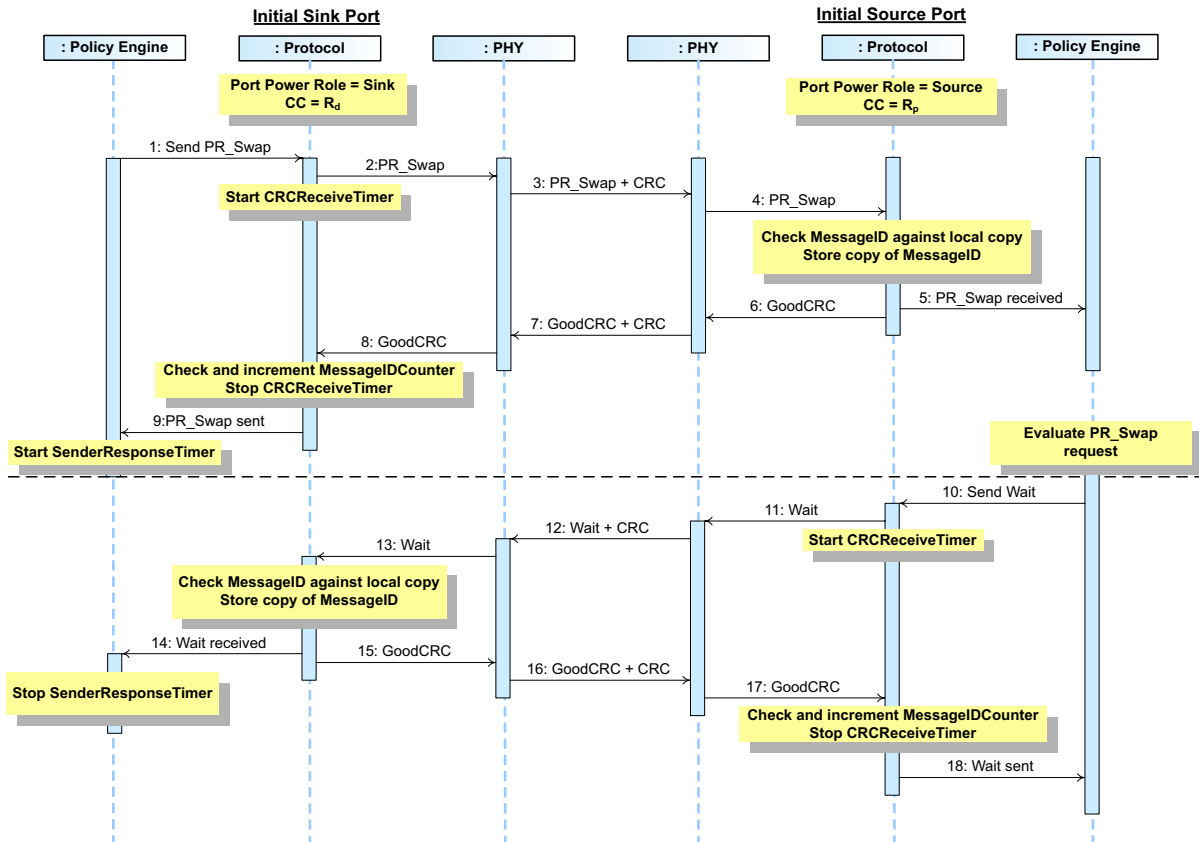
This is an example of a *Power Role Swap* operation, responded to with wait, initiated by a *Port* which initially, at the start of this *Message* sequence, is acting as a *Sink* and therefore has R_d pulled down on its *CC* wire.

There are several phases to the *Power Role Swap*:

- A *PR_Swap* Message is sent.
- A *Wait* Message in response to the *PR_Swap* Message.

[Figure 8.38, "Power Role Swap Sequence with wait Initiated by the Sink"](#) shows the Messages as they flow across the bus and within the devices.

Figure 8.38 Power Role Swap Sequence with wait Initiated by the Sink



[Table 8.65, "Steps for a Sink Initiated Power Role Swap with Wait Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.38, "Power Role Swap Sequence with wait Initiated by the Sink"](#) above.

Table 8.65 Steps for a Sink Initiated Power Role Swap with Wait Sequence

Step	Initial Sink Port	Initial Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a PR_Swap Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the PR_Swap Message. Starts CRCReceiveTimer .	PHY Layer receives the PR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the PR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received PR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the PR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the PR_Swap Message sent by the Sink and decides that it is able and willing to do the Power Role Swap but not at this time. It tells the Protocol Layer to form a Wait Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Wait Message.	PHY Layer appends a CRC and sends the Wait Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Wait Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.65 Steps for a Sink Initiated Power Role Swap with Wait Sequence

Step	Initial Sink Port	Initial Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Wait Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent

8.3.2.8 Fast Role Swap

This is an example of a successful *Fast Role Swap* operation initiated by a *Port* that is initially a *Source* and therefore has R_p pulled up on its *CC* wire and which has lost power and needs to get **vSafe5V** quickly. It does not include any subsequent *Power Negotiation* which is required in order to establish an *Explicit Contract* (see [Section 8.3.2.2, "Power Negotiation"](#)).

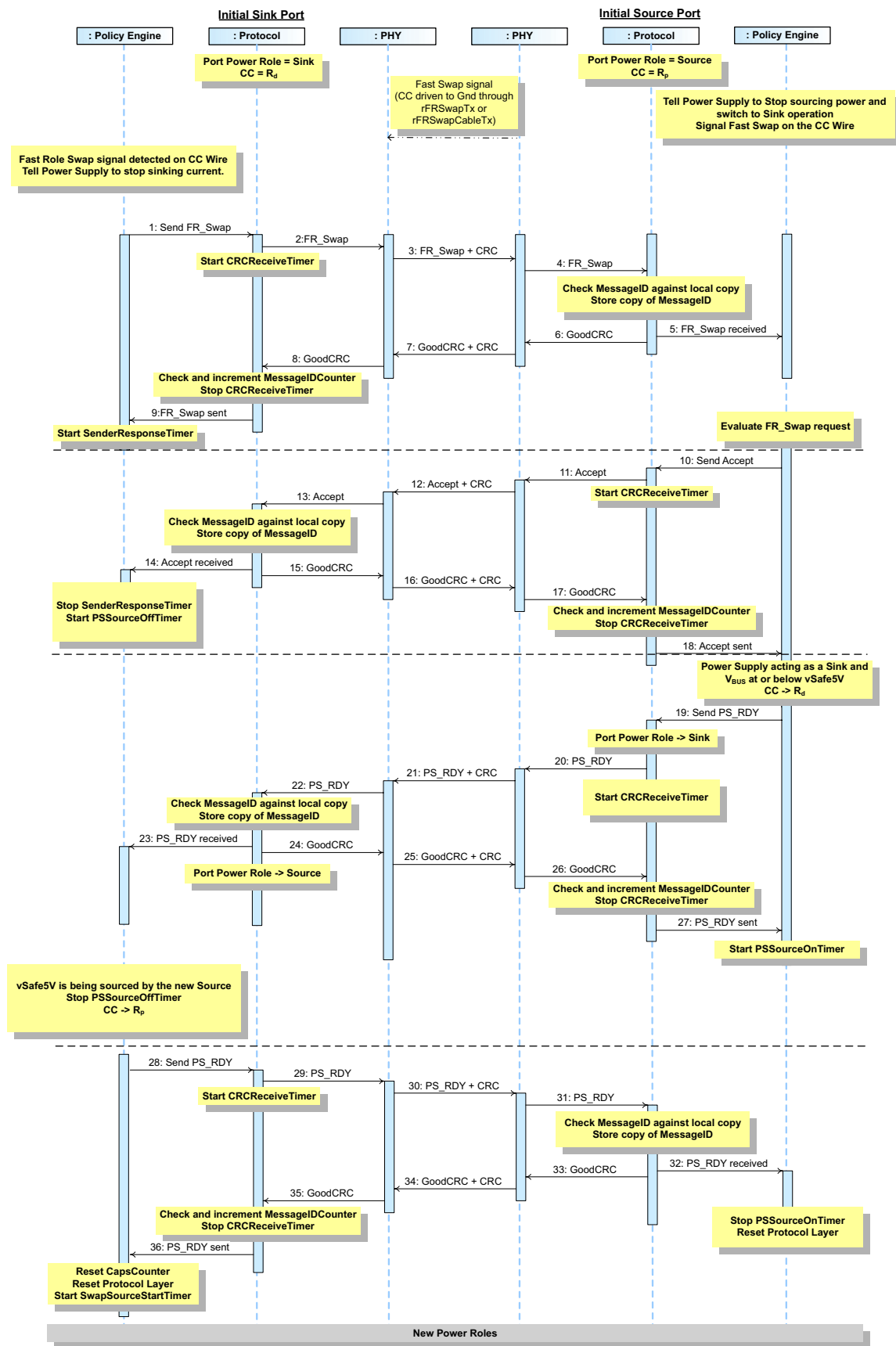
There are several distinct phases to the *Fast Role Swap Negotiation*:

- The *Initial Source* stops driving its power output which starts transitioning to **vSafe0V** and send the *Fast Role Swap Request* on the *CC* wire; these could occur in either order or simultaneously.
- The *Initial Sink* stops sinking power. At this point the *New Source* still has R_d asserted and the *New Sink* still has R_p asserted.
- An **FR_Swap** Message is sent by the *New Source* within **tFRSwapInit** of detecting the Fast Swap signal.
- An **Accept** Message is sent by the *New Sink* in response to the **FR_Swap** Message.
- The *New Sink* asserts R_d and sends a **PS_RDY** Message indicating that the voltage on *VBUS* is at or below **vSafe5V**.
- The *New Source* asserts R_p and sends a **PS_RDY** Message indicating that it is acting as a *Source* and is supplying **vSafe5V**.

Note: The *New Source* can start applying *VBUS* when *VBUS* is at or below **vSafe5V** (max) but will start driving *VBUS* to **vSafe5V** no later than **tSrcFRSwap** after detecting both the *Fast Role Swap Request* and that *VBUS* has dropped below **vSafe5V** (min).

[Figure 8.39, "Successful Fast Role Swap Sequence"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the *Fast Role Swap*.

Figure 8.39 Successful Fast Role Swap Sequence



[Table 8.66, "Steps for a Successful Fast Role Swap Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.39, "Successful Fast Role Swap Sequence"](#) above.

Table 8.66 Steps for a Successful Fast Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
1	<p>The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.</p> <p>The DPM detects Fast Swap on the CC wire and tells the power supply to stop sinking current.</p> <p>The Policy Engine directs the Protocol Layer to send an FR_Swap Message within tFRSwapInit of detecting the Fast Swap signal.</p>	<p>The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire.</p> <p>The DPM tells the Power Supply to stop sourcing power and switch to <i>Sink</i> operation.</p> <p>The DPM signals Fast Swap on the CC wire by driving CC to ground with a resistance of less than rFRSwapTx for at least tFRSwapTx.</p>
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the FR_Swap Message. Starts CRCReceiveTimer .	PHY Layer receives the FR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the PR_Swap Message to the Protocol Layer.
5		<p>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The Protocol Layer forwards the received FR_Swap Message information to the Policy Engine that consumes it.</p>
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	<p>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer.</p> <p>Protocol Layer informs the Policy Engine that the FR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer.</p>	
10		Policy Engine evaluates the PR_Swap Message sent by the Sink and decides that it is able and willing to do the Power Role Swap. It tells the Protocol Layer to form an Accept Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Accept Message.	PHY Layer appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	<p>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The Protocol Layer forwards the received PR_Swap Message information to the Policy Engine that consumes it.</p>	
14	The Policy Engine stops the SenderResponseTimer , starts the PSSourceOffTimer .	

Table 8.66 Steps for a Successful Fast Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Accept Message was successfully sent.
19		The <i>Policy Engine</i> determines its power supply is no longer supplying <i>V_{BUS}</i> and is acting as a Sink. The <i>Policy Engine</i> requests the <i>DPM</i> to assert the R_d pull down on the CC wire. The <i>Policy Engine</i> then directs the <i>Protocol Layer</i> to generate a PS_RDY Message, with the Port Power Role Messageset to Sink, to tell its Port Partner that it can begin to source <i>V_{BUS}</i> .
20		<i>Protocol Layer</i> sets the Port Power Role Message to Sink, creates the Message and passes to <i>PHY Layer</i> .
21	<i>PHY Layer</i> receives the PS_RDY Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the PS_RDY Message. Starts CRCReceiveTimer .
22	<i>PHY Layer</i> removes the CRC and forwards the PS_RDY Message to the <i>Protocol Layer</i> .	
23	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received PS_RDY Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the PSSourceOffTimer .	
24	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
25	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.
26		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
27		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the PS_RDY Message was successfully sent. <i>Policy Engine</i> starts PSSourceOnTimer .

Table 8.66 Steps for a Successful Fast Role Swap Sequence

Step	Initial Sink Port	Initial Source Port
28	<p>The <i>Policy Engine</i> directs the <i>DPM</i> to apply the R_p pull up.</p> <p>Note: At some point (either before or after receiving the <i>PS_RDY</i> Message) the <i>New Source</i> has applied <i>vSafe5V</i> no later than <i>tSrcFRSwap</i> after detecting the <i>Fast Role Swap Request</i> and that <i>VBUS</i> has dropped below <i>vSafe5V</i>.</p> <p><i>Policy Engine</i>, when its power supply is ready to supply power, tells the <i>Protocol Layer</i> to form a <i>PS_RDY</i> Message. The <i>Port Power Role</i> Message is set to <i>Source</i>.</p>	
29	<i>Protocol Layer</i> creates the <i>PS_RDY</i> Message and passes to <i>PHY Layer</i> .	
30	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>PS_RDY</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>PS_RDY</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
31		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>PS_RDY</i> Message to the <i>Protocol Layer</i> .
32		<p><i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received <i>PS_RDY</i> Message information to the <i>Policy Engine</i> that consumes it. The <i>Policy Engine</i> stops the <i>PSSourceOnTimer</i>.</p>
33		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
34	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message. The <i>Policy Engine</i> resets the <i>Protocol Layer</i> .
35	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> to the <i>Protocol Layer</i> .	
36	<p><i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i>.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>PS_RDY</i> Message was successfully sent. The <i>Policy Engine</i> resets the <i>CapsCounter</i>, resets the <i>Protocol Layer</i> and starts the <i>SwapSourceStartTimer</i> which must timeout before sending any <i>Source_Capabilities</i> Messages.</p>	
The <i>Fast Role Swap</i> is complete, the <i>Power Roles</i> have been reversed and the <i>Port Partners</i> are free to <i>Negotiate</i> for more power.		

[Table 8.67, "Steps for Data Role Swap, UFP operating as Sink initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.40, "Data Role Swap, UFP operating as Sink initiates"](#) above.

Table 8.67 Steps for Data Role Swap, UFP operating as Sink initiates

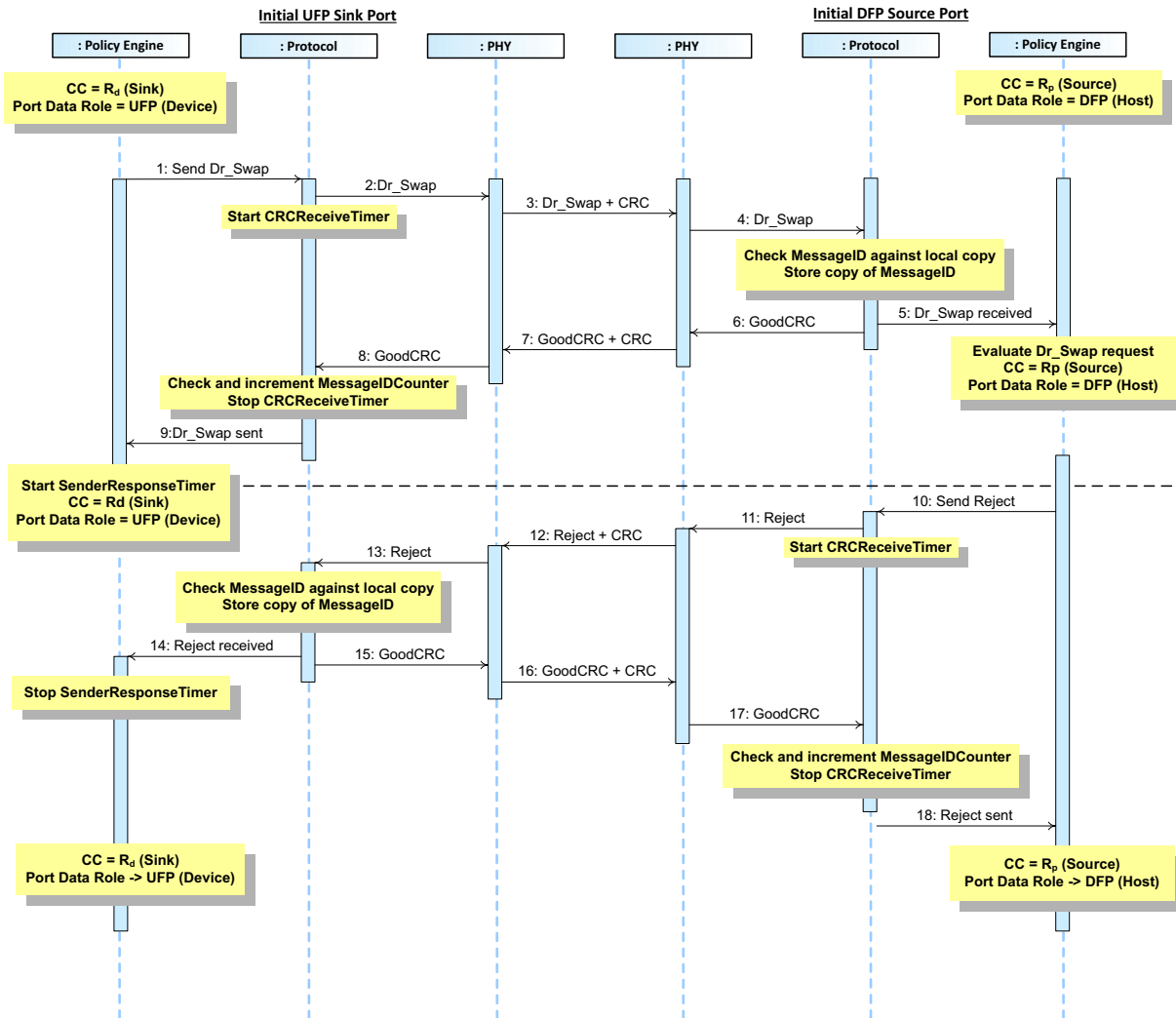
Step	Initial UFP Sink Port	Initial DFP Source Port
1	Port starts as a UFP (Device) operating as a Sink with R_d asserted and Port Data Role set to UFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .	Port starts as a DFP (Host) operating as Source with R_p asserted and Port Data Role set to DFP.
2	Protocol Layer creates the DR_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message .
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap. It tells the Protocol Layer to form an Accept Message .
11		Protocol Layer creates the Accept Message and passes to PHY Layer.
12	PHY Layer receives the Accept Message and checks the CRC to verify the Message.	PHY Layer appends a CRC and sends the Accept Message . Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Accept Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
16	PHY Layer appends a CRC and sends the GoodCRC Message .	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.67 Steps for Data Role Swap, UFP operating as Sink initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18	<p>The <i>Policy Engine</i> requests that <i>Data Role</i> is changed from <i>UFP (Device)</i> to <i>DFP (Host)</i>.</p> <p>The Power Delivery <i>Data Role</i> is now a <i>DFP (Host)</i>, with <i>Port Data Role</i> set to <i>DFP</i>, still operating as a <i>Sink</i> (R_d asserted).</p>	<p><i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i>. <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept</i> Message was successfully sent.</p> <p>The <i>Policy Engine</i> requests that the <i>Data Role</i> is changed to <i>UFP (Device)</i>, with <i>Port Data Role</i> set to <i>UFP</i> and continues supplying power as a <i>Source</i> (R_p asserted).</p>
The <i>Data Role Swap</i> is complete; the <i>Data Roles</i> have been reversed while maintaining the direction of power flow.		

Figure 8.41, "Rejected Data Role Swap, UFP operating as Sink initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Sink (R_d asserted), and a Port which is initially a DFP (Host) and a Source (R_p asserted). A Data Role Swap is initiated by the UFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) and the exchange of Data Roles is rejected.

Figure 8.41 Rejected Data Role Swap, UFP operating as Sink initiates



[Table 8.68, "Steps for Rejected Data Role Swap, UFP operating as Sink initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.41, "Rejected Data Role Swap, UFP operating as Sink initiates"](#) above.

Table 8.68 Steps for Rejected Data Role Swap, UFP operating as Sink initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
1	Port starts as a UFP (Device) operating as a Sink with R_d asserted and Port Data Role set to UFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message.	Port starts as a DFP (Host) operating as Source with R_p asserted and Port Data Role set to DFP.
2	Protocol Layer creates the DR_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the DR_Swap Message. Starts CRCReceiveTimer .	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the DR_Swap Message and decides that it is unable and unwilling to do the Data Role Swap. It tells the Protocol Layer to form a Reject Message.
11		Protocol Layer creates the Reject Message and passes to PHY Layer.
12	PHY Layer receives the Reject Message and checks the CRC to verify the Message.	PHY Layer appends a CRC and sends the Reject Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

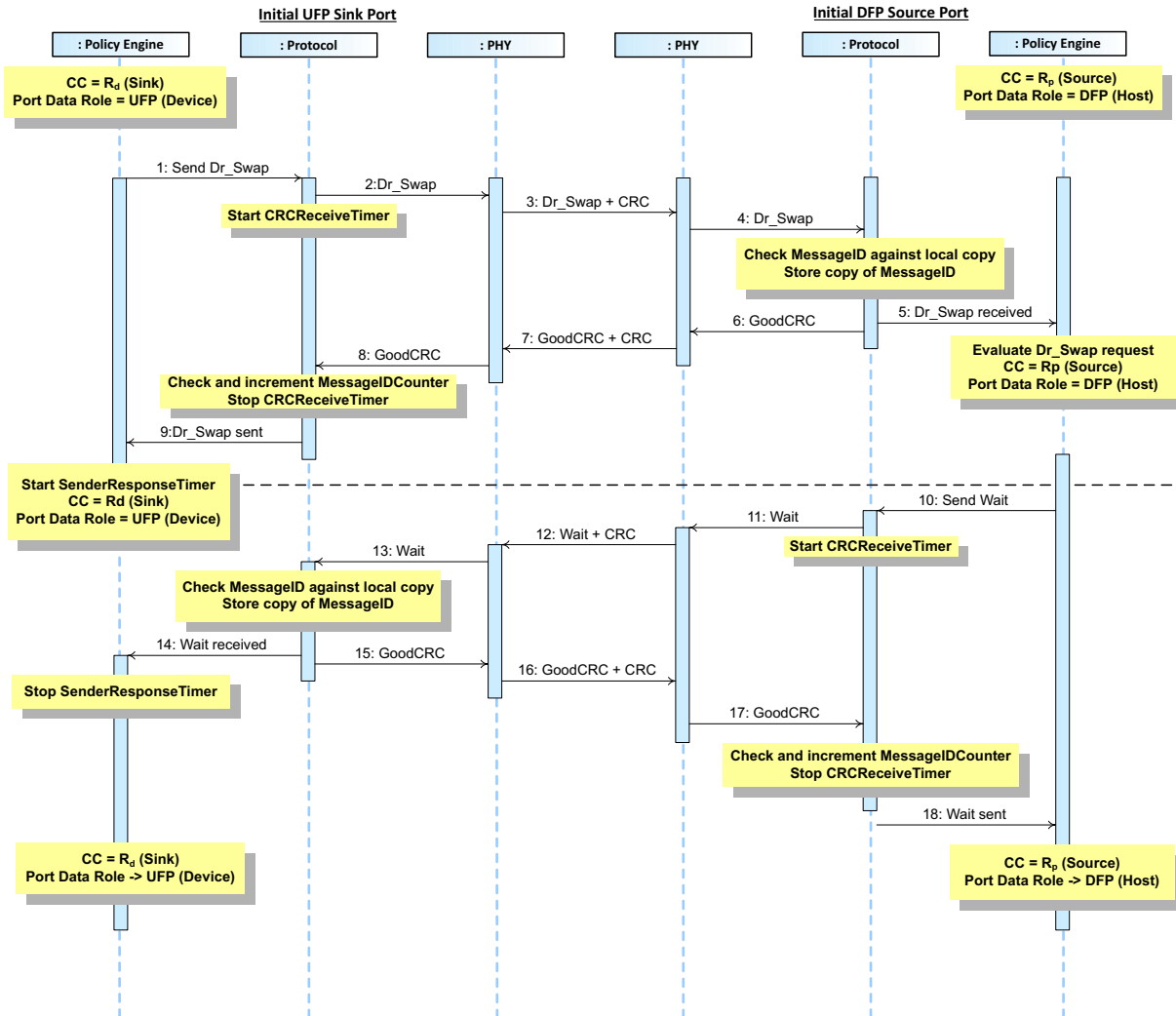
Table 8.68 Steps for Rejected Data Role Swap, UFP operating as Sink initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject Message</i> was successfully sent.

Data Role Swap, Initiated by UFP Operating as Sink (Wait)

Figure 8.42, "Data Role Swap with Wait, UFP operating as Sink initiates" shows an example sequence between a *Port*, which is initially a *UFP (Device)* and a *Sink* (R_d asserted), and a *Port* which is initially a *DFP (Host)* and a *Source* (R_p asserted). A *Data Role Swap* is initiated by the *UFP*. During the process the *Port Partners* maintain their operation as either a *Source* or a *Sink* (power and R_p/R_d remain constant) and the exchange of *Data Roles* is delayed with a wait.

Figure 8.42 Data Role Swap with Wait, UFP operating as Sink initiates



[Table 8.69, "Steps for Data Role Swap with Wait, UFP operating as Sink initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.42, "Data Role Swap with Wait, UFP operating as Sink initiates"](#) above.

Table 8.69 Steps for Data Role Swap with Wait, UFP operating as Sink initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
1	Port starts as a UFP (Device) operating as a Sink with R_d asserted and Port Data Role set to UFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .	Port starts as a DFP (Host) operating as Source with R_p asserted and Port Data Role set to DFP.
2	Protocol Layer creates the DR_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message .
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap but not at this time. It tells the Protocol Layer to form a Wait Message .
11		Protocol Layer creates the Wait Message and passes to PHY Layer.
12	PHY Layer receives the Wait Message and checks the CRC to verify the Message.	PHY Layer appends a CRC and sends the Wait Message . Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Wait Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.69 Steps for Data Role Swap with Wait, UFP operating as Sink initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent.

8.3.2.9.2

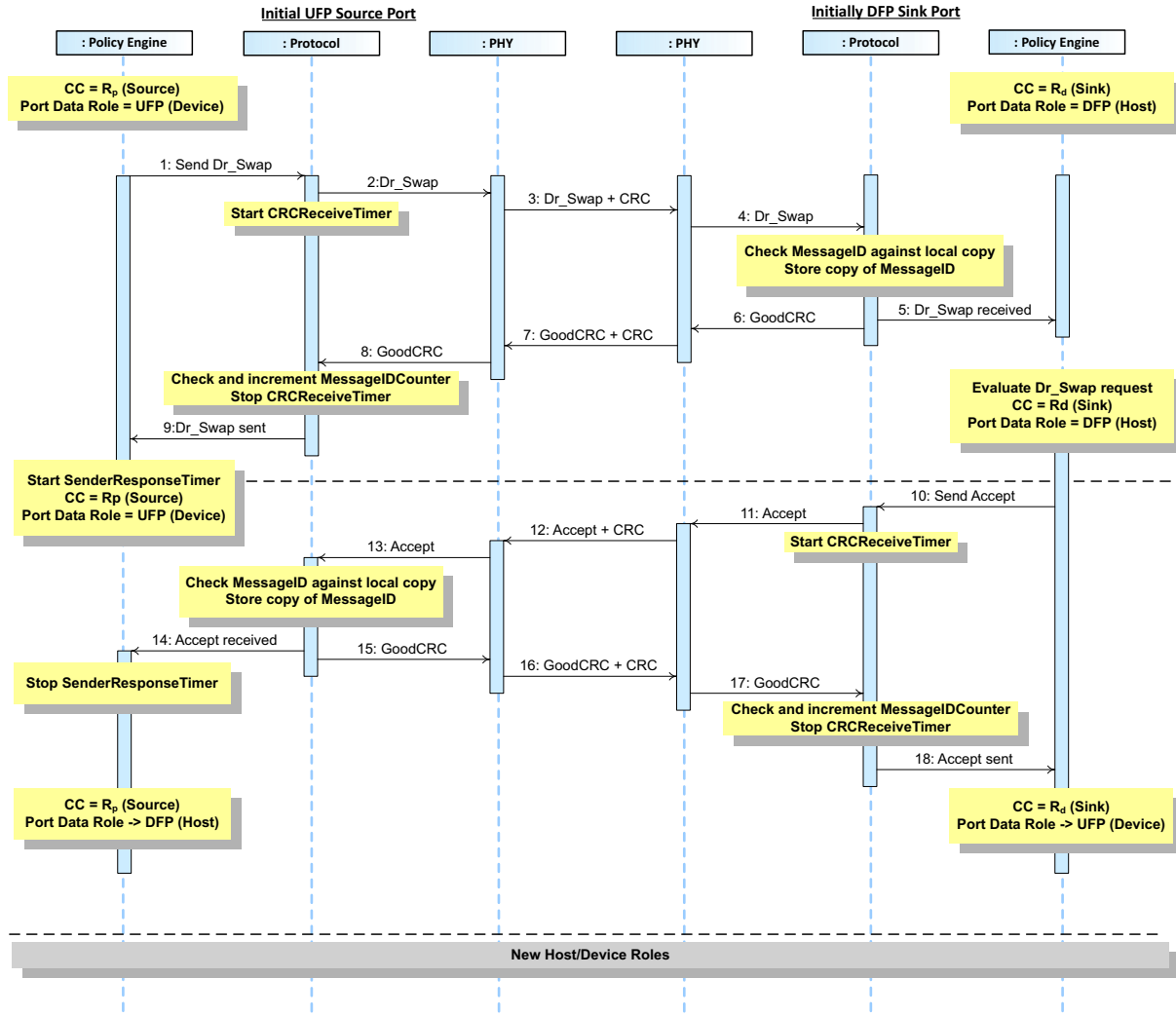
Data Role Swap, Initiated by UFP Operating as Source

8.3.2.9.2.1

Data Role Swap, Initiated by UFP Operating as Source (Accept)

[Figure 8.43, "Data Role Swap, UFP operating as Source initiates"](#) shows an example sequence between a *Port*, which is initially a *UFP (Device)* and a *Source* (R_p asserted), and a *Port* which is initially a *DFP (Host)* and a *Sink* (R_d asserted). A *Data Role Swap* is initiated by the *UFP*. During the process the *Port Partners* maintain their operation as either a *Source* or a *Sink* (power and R_p/R_d remain constant) but exchange *Data Roles* between *DFP (Host)* and *UFP (Device)*.

Figure 8.43 Data Role Swap, UFP operating as Source initiates



[Table 8.70, "Steps for Data Role Swap, UFP operating as Source initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.43, "Data Role Swap, UFP operating as Source initiates"](#) above.

Table 8.70 Steps for Data Role Swap, UFP operating as Source initiates

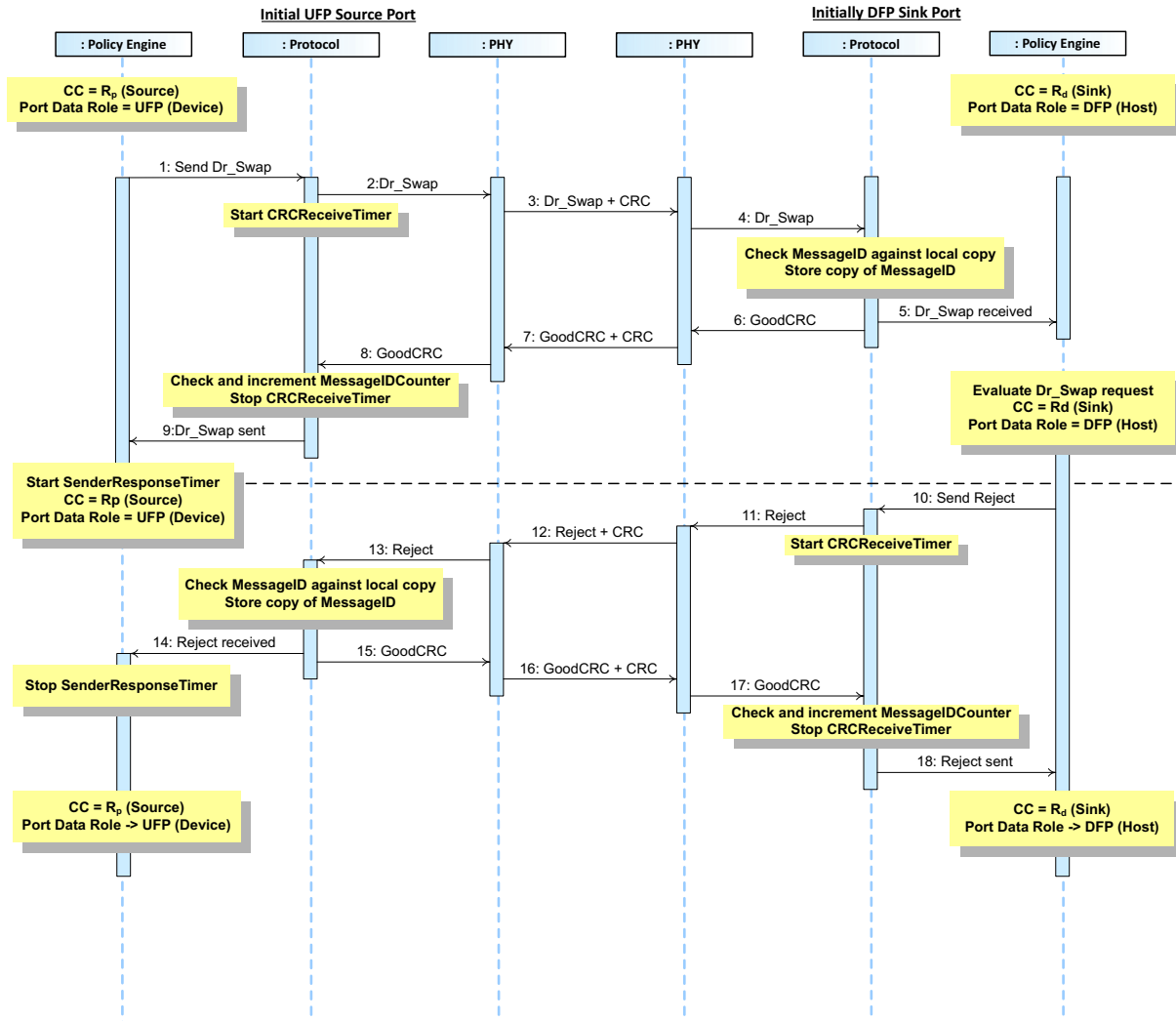
Step	Initial UFP Source Port	Initial DFP Sink Port
1	Port starts as a UFP (Device) operating as Source with R_p asserted and Port Data Role set to UFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .	Port starts as a DFP (Host) operating as a Sink with R_d asserted and Port Data Role set to DFP.
2	Protocol Layer creates the DR_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message .
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap. It tells the Protocol Layer to form an Accept Message .
11		Protocol Layer creates the Accept Message and passes to PHY Layer.
12	PHY Layer receives the Accept Message and checks the CRC to verify the Message.	PHY Layer appends a CRC and sends the Accept Message . Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Accept Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
16	PHY Layer appends a CRC and sends the GoodCRC Message .	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.70 Steps for Data Role Swap, UFP operating as Source initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18	<p>The <i>Policy Engine</i> requests that <i>Data Role</i> is changed from <i>UFP (Device)</i> to <i>DFP (Host)</i>.</p> <p>The Power Delivery <i>Data Role</i> is now a <i>DFP (Host)</i>, and Port Data Role set to <i>DFP</i> and continues supplying power as a <i>Source</i> (R_p asserted).</p>	<p><i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Accept Message was successfully sent. The <i>Policy Engine</i> requests that the <i>Data Role</i> is changed to <i>UFP (Device)</i>, with Port Data Role set to <i>UFP</i> and still operating as a <i>Sink</i> (R_p asserted).</p>
The <i>Data Role Swap</i> is complete; the <i>Data Roles</i> have been reversed while maintaining the direction of power flow.		

Figure 8.44, "Rejected Data Role Swap, UFP operating as Source initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Source (R_p asserted), and a Port which is initially a DFP (Host) and a Sink (R_d asserted). A Data Role Swap is initiated by the UFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) and the exchange of Data Roles is rejected.

Figure 8.44 Rejected Data Role Swap, UFP operating as Source initiates



[Table 8.71, "Steps for Rejected Data Role Swap, UFP operating as Source initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.44, "Rejected Data Role Swap, UFP operating as Source initiates"](#) above.

Table 8.71 Steps for Rejected Data Role Swap, UFP operating as Source initiates

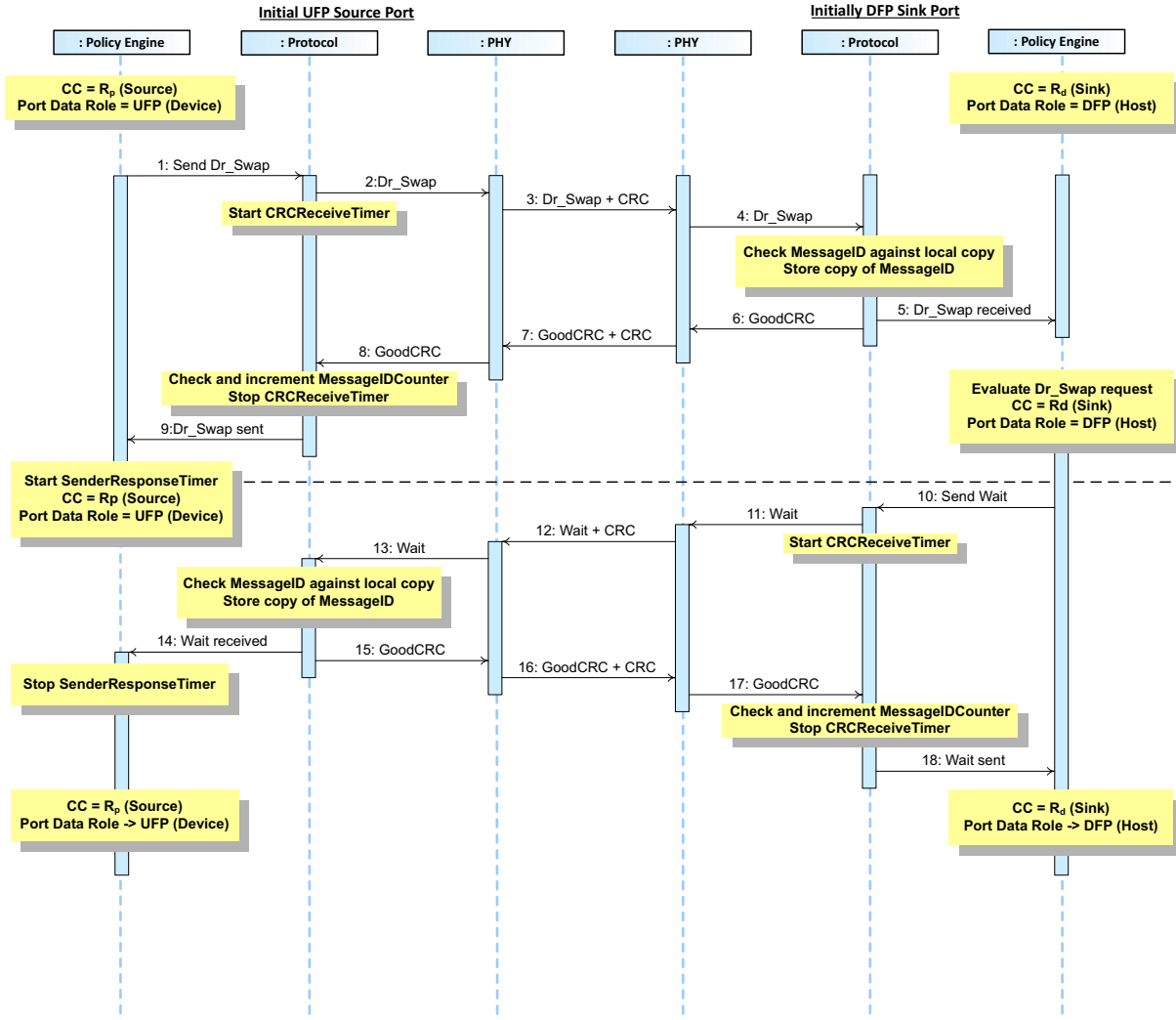
Step	Initial UFP Source Port	Initial DFP Sink Port
1	Port starts as a UFP (Device) operating as Source with R_p asserted and Port Data Role set to UFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .	Port starts as a DFP (Host) operating as a Sink with R_d asserted and Port Data Role set to DFP.
2	Protocol Layer creates the DR_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message .
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the DR_Swap Message and decides that it is unable and unwilling to do the Data Role Swap. It tells the Protocol Layer to form a Reject Message .
11		Protocol Layer creates the Reject Message and passes to PHY Layer.
12	PHY Layer receives the Reject Message and checks the CRC to verify the Message.	PHY Layer appends a CRC and sends the Reject Message . Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.71 Steps for Rejected Data Role Swap, UFP operating as Source initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject Message</i> was successfully sent.

Figure 8.45, "Data Role Swap with Wait, UFP operating as Source initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Source (R_p asserted), and a Port which is initially a DFP (Host) and a Sink (R_d asserted). A Data Role Swap is initiated by the UFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) and the exchange of Data Roles is delayed with a wait.

Figure 8.45 Data Role Swap with Wait, UFP operating as Source initiates



[Table 8.72, "Steps for Data Role Swap with Wait, UFP operating as Source initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.45, "Data Role Swap with Wait, UFP operating as Source initiates"](#) above.

Table 8.72 Steps for Data Role Swap with Wait, UFP operating as Source initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
1	Port starts as a UFP (Device) operating as Source with R_p asserted and Port Data Role set to UFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .	Port starts as a DFP (Host) operating as a Sink with R_d asserted and Port Data Role set to DFP.
2	Protocol Layer creates the DR_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message .
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap but not at this time. It tells the Protocol Layer to form a Wait Message .
11		Protocol Layer creates the Wait Message and passes to PHY Layer.
12	PHY Layer receives the Wait Message and checks the CRC to verify the Message.	PHY Layer appends a CRC and sends the Wait Message . Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Wait Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.72 Steps for Data Role Swap with Wait, UFP operating as Source initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent.

8.3.2.9.3

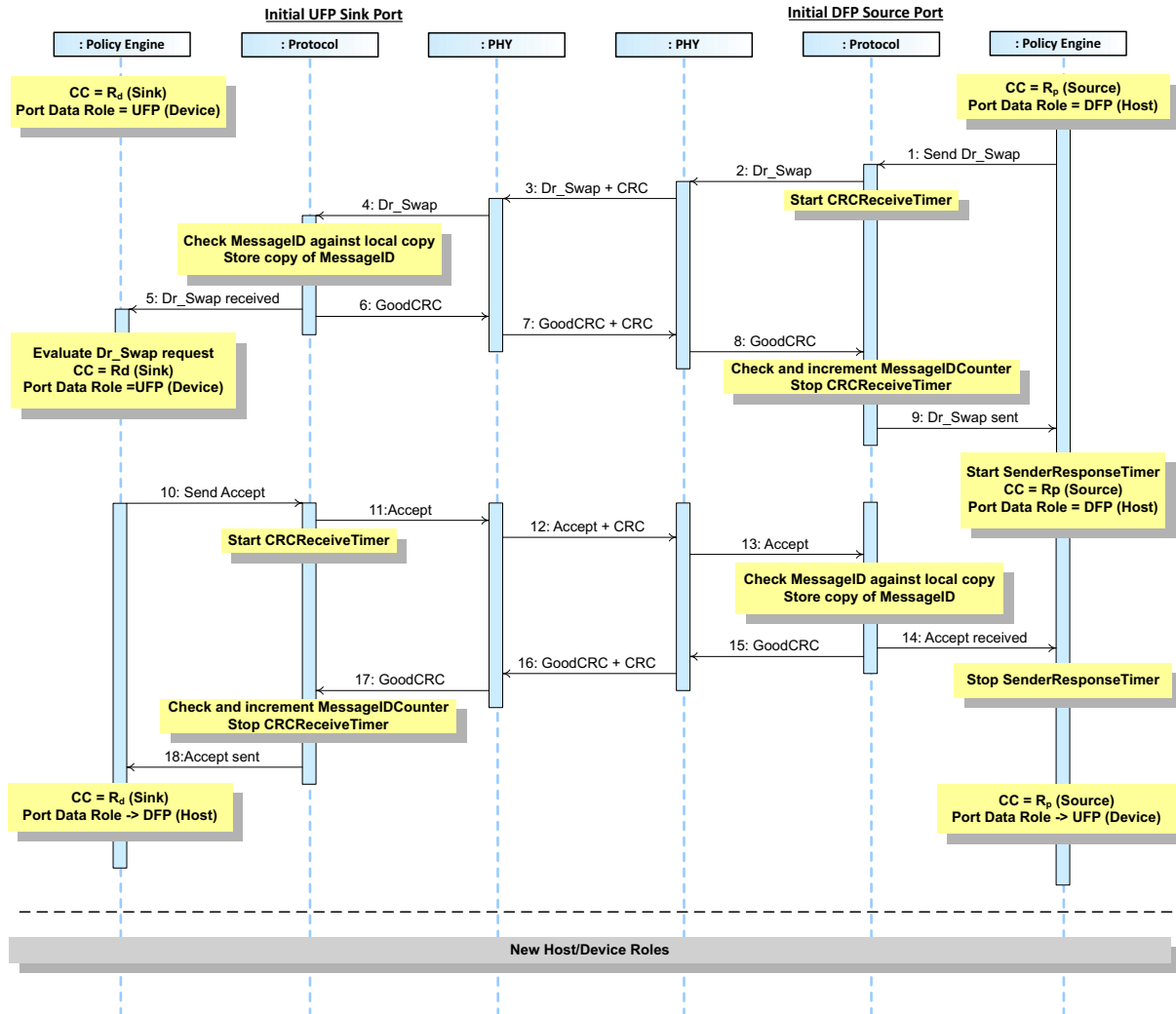
Data Role Swap, Initiated by DFP Operating as Source

8.3.2.9.3.1

Data Role Swap, Initiated by DFP Operating as Source (Accept)

Figure 8.46, "Data Role Swap, DFP operating as Source initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Sink (R_d asserted), and a Port which is initially a DFP and a Source (R_p asserted). A Data Role Swap is initiated by the DFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) but exchange Data Roles between DFP (Host) and UFP (Device).

Figure 8.46 Data Role Swap, DFP operating as Source initiates



[Table 8.73, "Steps for Data Role Swap, DFP operating as Source initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.46, "Data Role Swap, DFP operating as Source initiates"](#) above.

Table 8.73 Steps for Data Role Swap, DFP operating as Source initiates

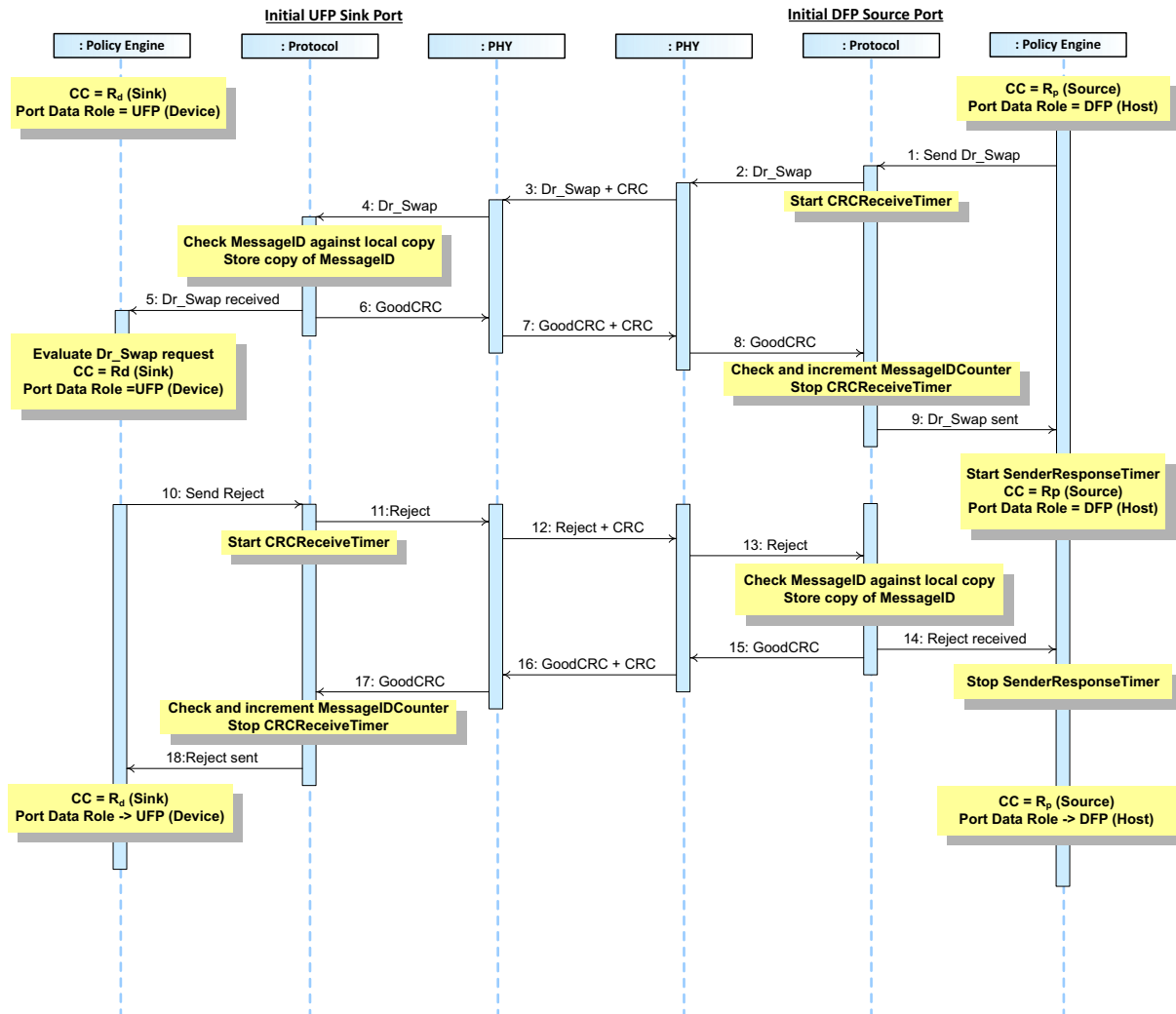
Step	Initial UFP Sink Port	Initial DFP Source Port
1	Port starts as a UFP (Device) operating as a Sink with R_d asserted and Port Data Role set to UFP.	Port starts as a DFP (Host) operating as Source with R_p asserted and Port Data Role set to DFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .
2		Protocol Layer creates the DR_Swap Message and passes to PHY Layer.
3	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .
4	PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.	
5	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.	
6	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
7	PHY Layer appends CRC and sends the GoodCRC Message .	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.
8		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
9		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .
10	Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap. It tells the Protocol Layer to form an Accept Message .	
11	Protocol Layer creates the Accept Message and passes to PHY Layer.	
12	PHY Layer appends a CRC and sends the Accept Message . Starts CRCReceiveTimer .	PHY Layer receives the Accept Message and checks the CRC to verify the Message.
13		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Accept Message information to the Policy Engine that consumes it.
14		The Policy Engine stops the SenderResponseTimer .
15		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
16	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the GoodCRC Message .

Table 8.73 Steps for Data Role Swap, DFP operating as Source initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
17	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept Message</i> was successfully sent. The <i>Policy Engine</i> requests that the <i>Data Role</i> is changed to <i>DFP (Host)</i> , with <i>Port Data Role</i> set to <i>DFP</i> , still operating as a <i>Sink</i> (R_d asserted).	<p>The <i>Policy Engine</i> requests that <i>Data Role</i> is changed from <i>DFP (Host)</i> to <i>UFP (Device)</i>.</p> <p>The <i>Power Delivery Data Role</i> is now a <i>UFP (Device)</i>, with <i>Port Data Role</i> set to <i>UFP</i> and continues supplying power as a <i>Source</i> (R_p asserted).</p>
The <i>Data Role Swap</i> is complete; the <i>Data Roles</i> have been reversed while maintaining the direction of power flow.		

Figure 8.47, "Rejected Data Role Swap, DFP operating as Source initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Sink (R_d asserted), and a Port which is initially a DFP and a Source (R_p asserted). A Data Role Swap is initiated by the DFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) and the exchange of Data Roles is rejected.

Figure 8.47 Rejected Data Role Swap, DFP operating as Source initiates



[Table 8.74, "Steps for Rejected Data Role Swap, DFP operating as Source initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.47, "Rejected Data Role Swap, DFP operating as Source initiates"](#) above.

Table 8.74 Steps for Rejected Data Role Swap, DFP operating as Source initiates

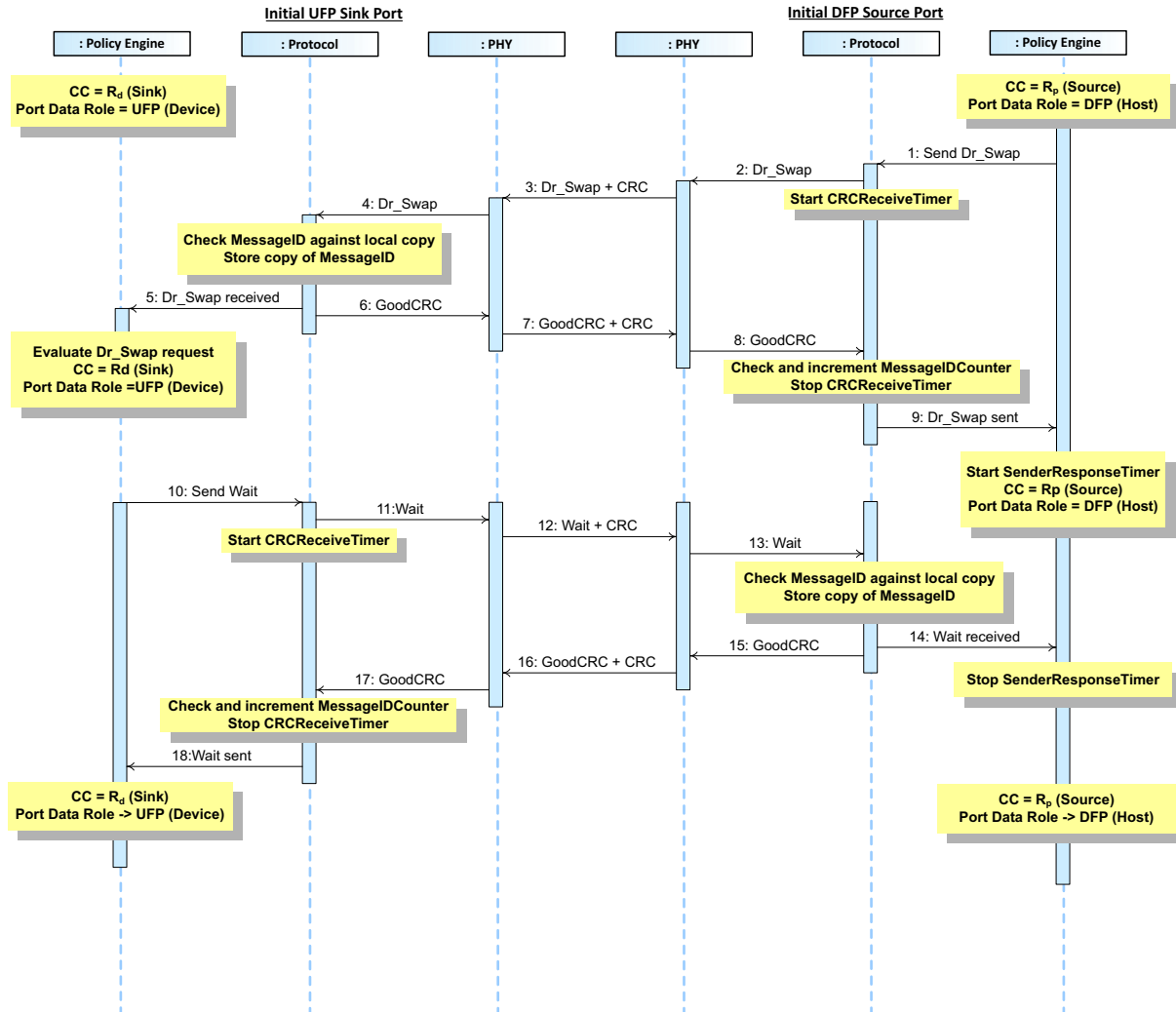
Step	Initial UFP Sink Port	Initial DFP Source Port
1	Port starts as a UFP (Device) operating as a Sink with R_d asserted and Port Data Role set to UFP.	Port starts as a DFP (Host) operating as Source with R_p asserted and Port Data Role set to DFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .
2		Protocol Layer creates the DR_Swap Message and passes to PHY Layer.
3	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .
4	PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.	
5	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.	
6	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
7	PHY Layer appends CRC and sends the GoodCRC Message .	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.
8		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
9		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .
10	Policy Engine evaluates the DR_Swap Message and decides that it is unable and unwilling to do the Data Role Swap. It tells the Protocol Layer to form an Reject Message .	
11	Protocol Layer creates the Reject Message and passes to PHY Layer.	
12	PHY Layer appends a CRC and sends the Reject Message . Starts CRCReceiveTimer .	PHY Layer receives the Reject Message and checks the CRC to verify the Message.
13		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.
14		The Policy Engine stops the SenderResponseTimer .
15		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.

Table 8.74 Steps for Rejected Data Role Swap, DFP operating as Source initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
16	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.
17	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Reject Message was successfully sent.	

Figure 8.48, "Data Role Swap with Wait, DFP operating as Source initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Sink (R_d asserted), and a Port which is initially a DFP and a Source (R_p asserted). A Data Role Swap is initiated by the DFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) and the exchange of Data Roles is delayed by wait.

Figure 8.48 Data Role Swap with Wait, DFP operating as Source initiates



[Table 8.75, "Steps for Data Role Swap with Wait, DFP operating as Source initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.48, "Data Role Swap with Wait, DFP operating as Source initiates"](#) above.

Table 8.75 Steps for Data Role Swap with Wait, DFP operating as Source initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
1	Port starts as a UFP (Device) operating as a Sink with R_d asserted and Port Data Role set to UFP.	Port starts as a DFP (Host) operating as Source with R_p asserted and Port Data Role set to DFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message .
2		Protocol Layer creates the DR_Swap Message and passes to PHY Layer.
3	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the DR_Swap Message . Starts CRCReceiveTimer .
4	PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.	
5	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.	
6	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
7	PHY Layer appends CRC and sends the GoodCRC Message .	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.
8		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
9		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .
10	Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap but not at this time. It tells the Protocol Layer to form an Wait Message .	
11	Protocol Layer creates the Wait Message and passes to PHY Layer.	
12	PHY Layer appends a CRC and sends the Wait Message . Starts CRCReceiveTimer .	PHY Layer receives the Wait Message and checks the CRC to verify the Message.
13		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Wait Message information to the Policy Engine that consumes it.
14		The Policy Engine stops the SenderResponseTimer .
15		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
16	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the GoodCRC Message .

Table 8.75 Steps for Data Role Swap with Wait, DFP operating as Source initiates

Step	Initial UFP Sink Port	Initial DFP Source Port
17	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait</i> Message was successfully sent.	

8.3.2.9.4

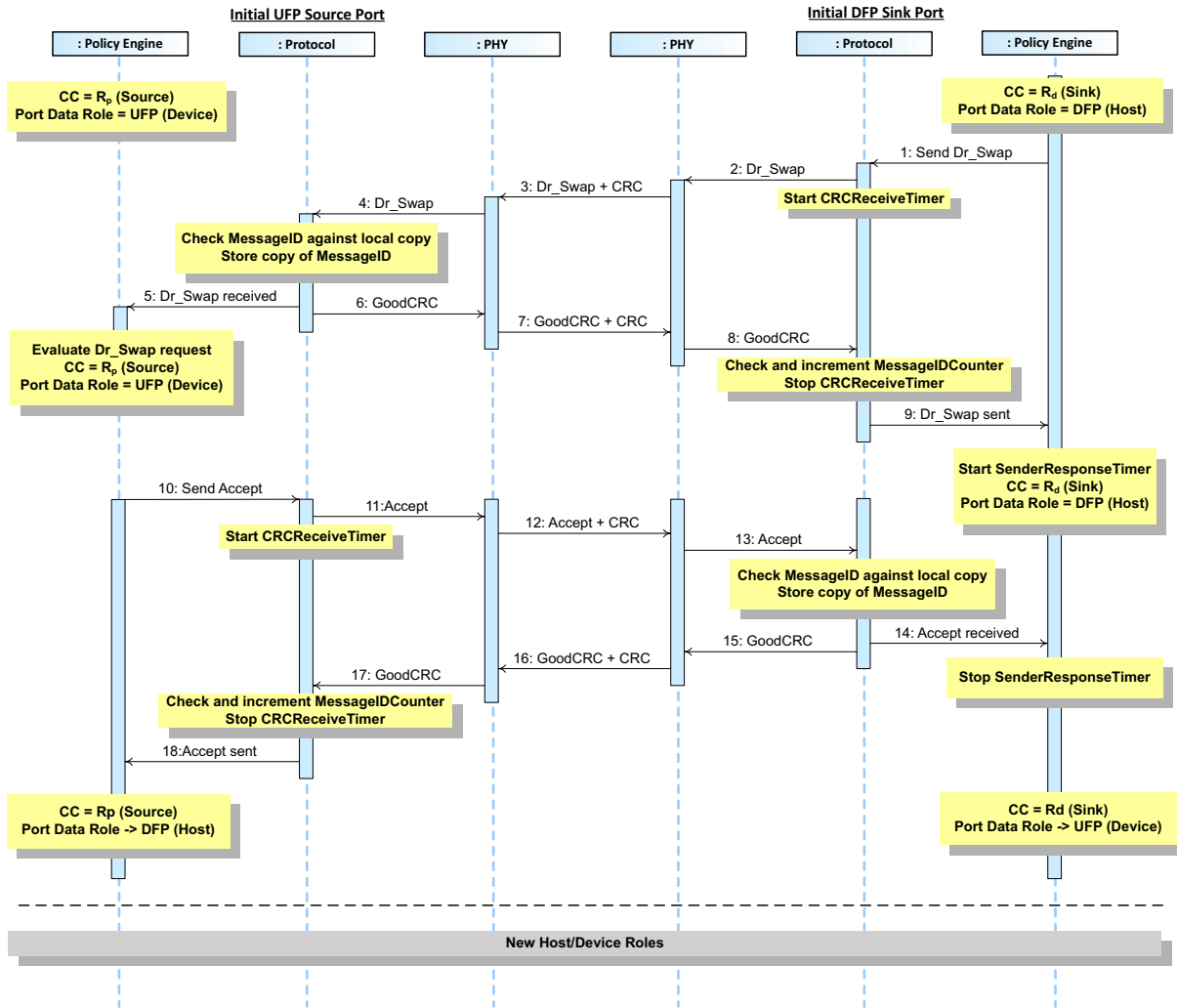
Data Role Swap, Initiated by DFP Operating as Sink

8.3.2.9.4.1

Data Role Swap, Initiated by DFP Operating as Sink (Accept)

Figure 8.49, "Data Role Swap, DFP operating as Sink initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Source (R_p asserted), and a Port which is initially a DFP (Host) and a Sink (R_d asserted). A Data Role Swap is initiated by the DFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) but exchange Data Roles between DFP (Host) and UFP (Device).

Figure 8.49 Data Role Swap, DFP operating as Sink initiates



[Table 8.76, "Steps for Data Role Swap, DFP operating as Sink initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.49, "Data Role Swap, DFP operating as Sink initiates"](#) above.

Table 8.76 Steps for Data Role Swap, DFP operating as Sink initiates

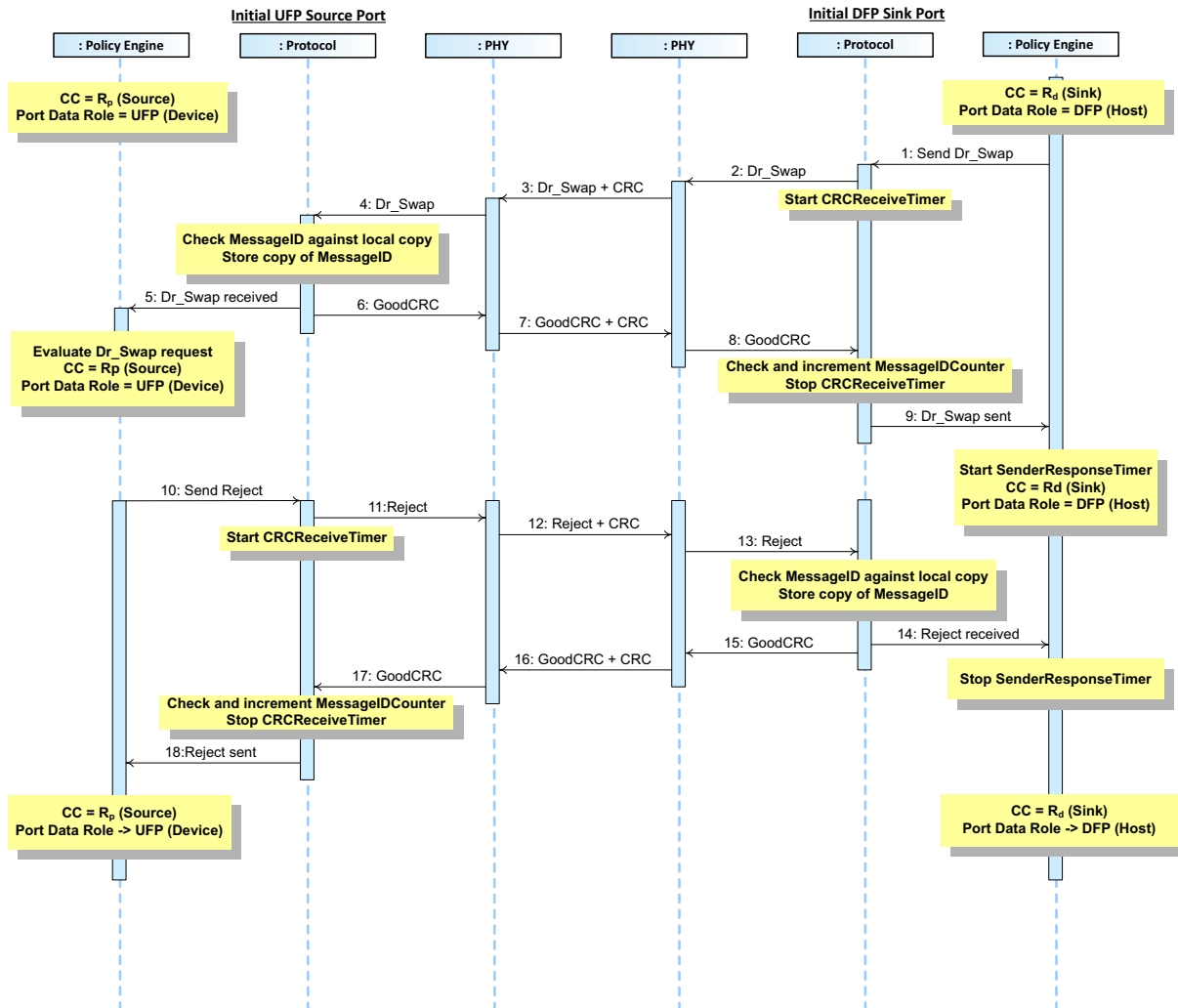
Step	Initial UFP Source Port	Initial DFP Sink Port
1	Port starts as a UFP (Device) operating as Source with R_p asserted and Port Data Role set to UFP.	Port starts as a DFP (Host) operating as a Sink with R_d asserted and Port Data Role set to DFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message.
2		Protocol Layer creates the DR_Swap Message and passes to PHY Layer.
3	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the DR_Swap Message. Starts CRCReceiveTimer .
4	PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.	
5	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.	
6	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
7	PHY Layer appends CRC and sends the GoodCRC Message.	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.
8		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
9		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .
10	Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap. It tells the Protocol Layer to form an Accept Message.	
11	Protocol Layer creates the Accept Message and passes to PHY Layer.	
12	PHY Layer appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .	PHY Layer receives the Accept Message and checks the CRC to verify the Message.
13		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Accept Message information to the Policy Engine that consumes it.
14		The Policy Engine stops the SenderResponseTimer .
15		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
16	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the GoodCRC Message.

Table 8.76 Steps for Data Role Swap, DFP operating as Sink initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
17	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept</i> Message was successfully sent. The <i>Policy Engine</i> requests that the <i>Data Role</i> is changed to <i>DFP (Host)</i> , with <i>Port Data Role</i> set to <i>DFP</i> and continues supplying power as a <i>Source</i> (R_p asserted).	The <i>Policy Engine</i> requests that <i>Data Role</i> is changed from <i>DFP (Host)</i> to <i>UFP (Device)</i> . The Power Delivery <i>Data Role</i> is now a <i>UFP (Device)</i> , with <i>Port Data Role</i> set to <i>UFP</i> , still operating as a <i>Sink</i> (R_d asserted).
The <i>Data Role Swap</i> is complete; the <i>Data Roles</i> have been reversed while maintaining the direction of power flow.		

Figure 8.50, "Rejected Data Role Swap, DFP operating as Sink initiates" shows an example sequence between a *Port*, which is initially a *UFP* (*Device*) and a *Source* (R_p asserted), and a *Port* which is initially a *DFP* (*Host*) and a *Sink* (R_d asserted). A *Data Role Swap* is initiated by the *DFP*. During the process the *Port Partners* maintain their operation as either a *Source* or a *Sink* (power and R_p/R_d remain constant) and the exchange of *Data Roles* is rejected.

Figure 8.50 Rejected Data Role Swap, DFP operating as Sink initiates



[Table 8.77, "Steps for Rejected Data Role Swap, DFP operating as Sink initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.50, "Rejected Data Role Swap, DFP operating as Sink initiates"](#) above.

Table 8.77 Steps for Rejected Data Role Swap, DFP operating as Sink initiates

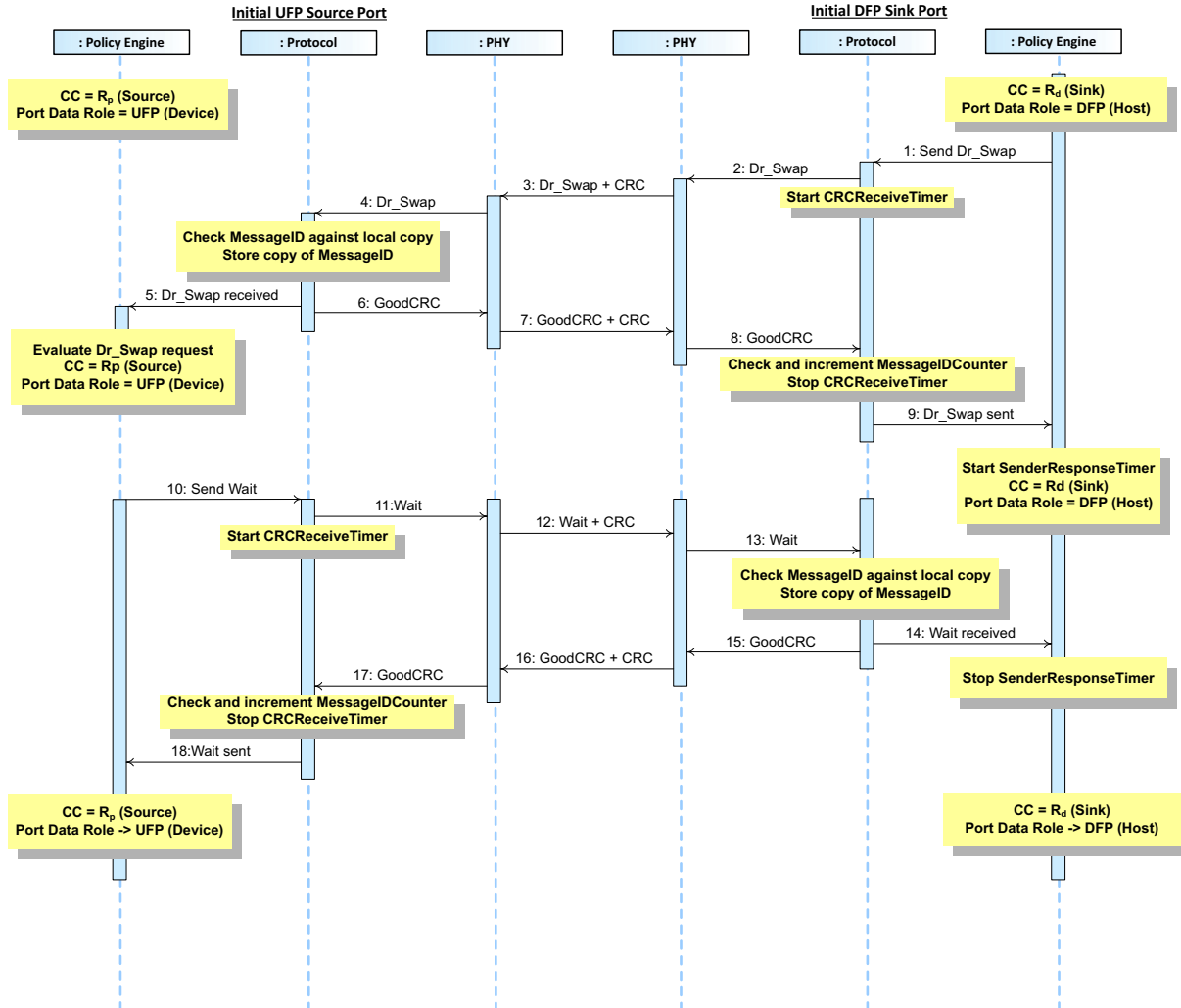
Step	Initial UFP Source Port	Initial DFP Sink Port
1	Port starts as a UFP (Device) operating as Source with R_p asserted and Port Data Role set to UFP.	Port starts as a DFP (Host) operating as a Sink with R_d asserted and Port Data Role set to DFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message.
2		Protocol Layer creates the DR_Swap Message and passes to PHY Layer.
3	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the DR_Swap Message. Starts CRCReceiveTimer .
4	PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.	
5	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.	
6	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
7	PHY Layer appends CRC and sends the GoodCRC Message.	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.
8		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
9		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .
10	Policy Engine evaluates the DR_Swap Message and decides that it is unable and unwilling to do the Data Role Swap. It tells the Protocol Layer to form an Reject Message.	
11	Protocol Layer creates the Reject Message and passes to PHY Layer.	
12	PHY Layer appends a CRC and sends the Reject Message. Starts CRCReceiveTimer .	PHY Layer receives the Reject Message and checks the CRC to verify the Message.
13		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.
14		The Policy Engine stops the SenderResponseTimer .
15		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.

Table 8.77 Steps for Rejected Data Role Swap, DFP operating as Sink initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
16	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.
17	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
18	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Reject Message was successfully sent.	

Figure 8.51, "Data Role Swap with Wait, DFP operating as Sink initiates" shows an example sequence between a Port, which is initially a UFP (Device) and a Source (R_p asserted), and a Port which is initially a DFP (Host) and a Sink (R_d asserted). A Data Role Swap is initiated by the DFP. During the process the Port Partners maintain their operation as either a Source or a Sink (power and R_p/R_d remain constant) and the exchange of Data Roles is delayed with a wait.

Figure 8.51 Data Role Swap with Wait, DFP operating as Sink initiates



[Table 8.78, "Steps for Data Role Swap with Wait, DFP operating as Sink initiates"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.51, "Data Role Swap with Wait, DFP operating as Sink initiates"](#) above.

Table 8.78 Steps for Data Role Swap with Wait, DFP operating as Sink initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
1	Port starts as a UFP (Device) operating as Source with R_p asserted and Port Data Role set to UFP.	Port starts as a DFP (Host) operating as a Sink with R_d asserted and Port Data Role set to DFP. The Policy Engine directs the Protocol Layer to send a DR_Swap Message.
2		Protocol Layer creates the DR_Swap Message and passes to PHY Layer.
3	PHY Layer receives the DR_Swap Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the DR_Swap Message. Starts CRCReceiveTimer .
4	PHY Layer removes the CRC and forwards the DR_Swap Message to the Protocol Layer.	
5	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received DR_Swap Message information to the Policy Engine that consumes it.	
6	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
7	PHY Layer appends CRC and sends the GoodCRC Message.	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.
8		PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.
9		Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the DR_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .
10	Policy Engine evaluates the DR_Swap Message and decides that it is able and willing to do the Data Role Swap but not at this time. It tells the Protocol Layer to form a Wait Message.	
11	Protocol Layer creates the Wait Message and passes to PHY Layer.	
12	PHY Layer appends a CRC and sends the Wait Message. Starts CRCReceiveTimer .	PHY Layer receives the Wait Message and checks the CRC to verify the Message.
13		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.
14		The Policy Engine stops the SenderResponseTimer .
15		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.

Table 8.78 Steps for Data Role Swap with Wait, DFP operating as Sink initiates

Step	Initial UFP Source Port	Initial DFP Sink Port
16	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.
17	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
18	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Wait Message was successfully sent.	

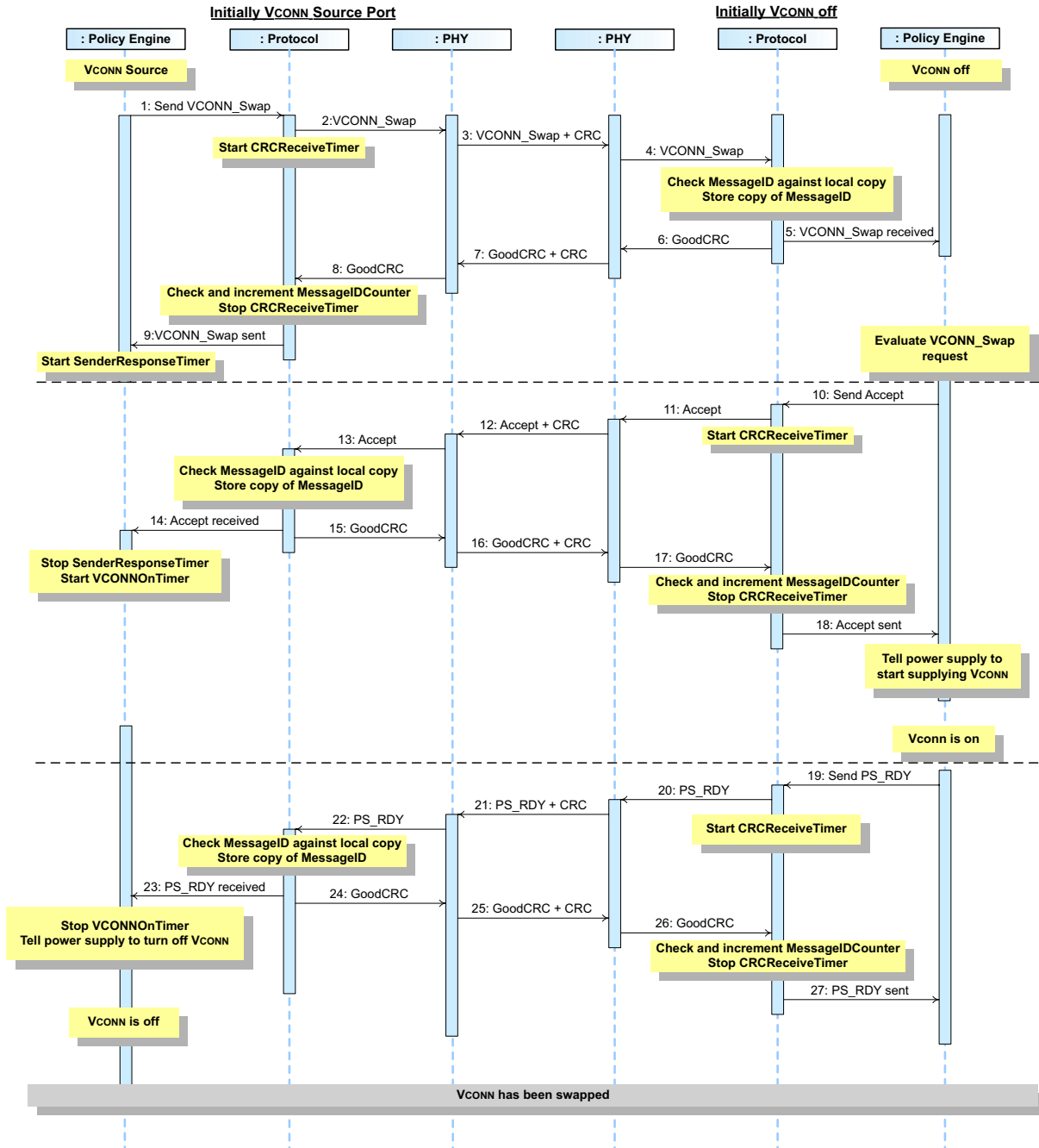
8.3.2.10 VCONN Swap

8.3.2.10.1 VCONN Source Swap, initiated by VCONN Source

8.3.2.10.1.1 VCONN Source Swap, initiated by VCONN Source (Accept)

Figure 8.52, "Successful VCONN Source Swap, initiated by VCONN Source" shows an example sequence where the VCONN Source and tells its Port Partner to supply VCONN. During the process the Port Partners, keep their Power Role as Source or Sink, maintain their operation as either a Source or a Sink (power remains constant) but exchange the VCONN Source role.

Figure 8.52 Successful VCONN Source Swap, initiated by VCONN Source



[Table 8.79, "Steps for Source to Sink VCONN Source Swap"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.52, "Successful VCONN Source Swap, initiated by VCONN Source"](#) above.

Table 8.79 Steps for Source to Sink VCONN Source Swap

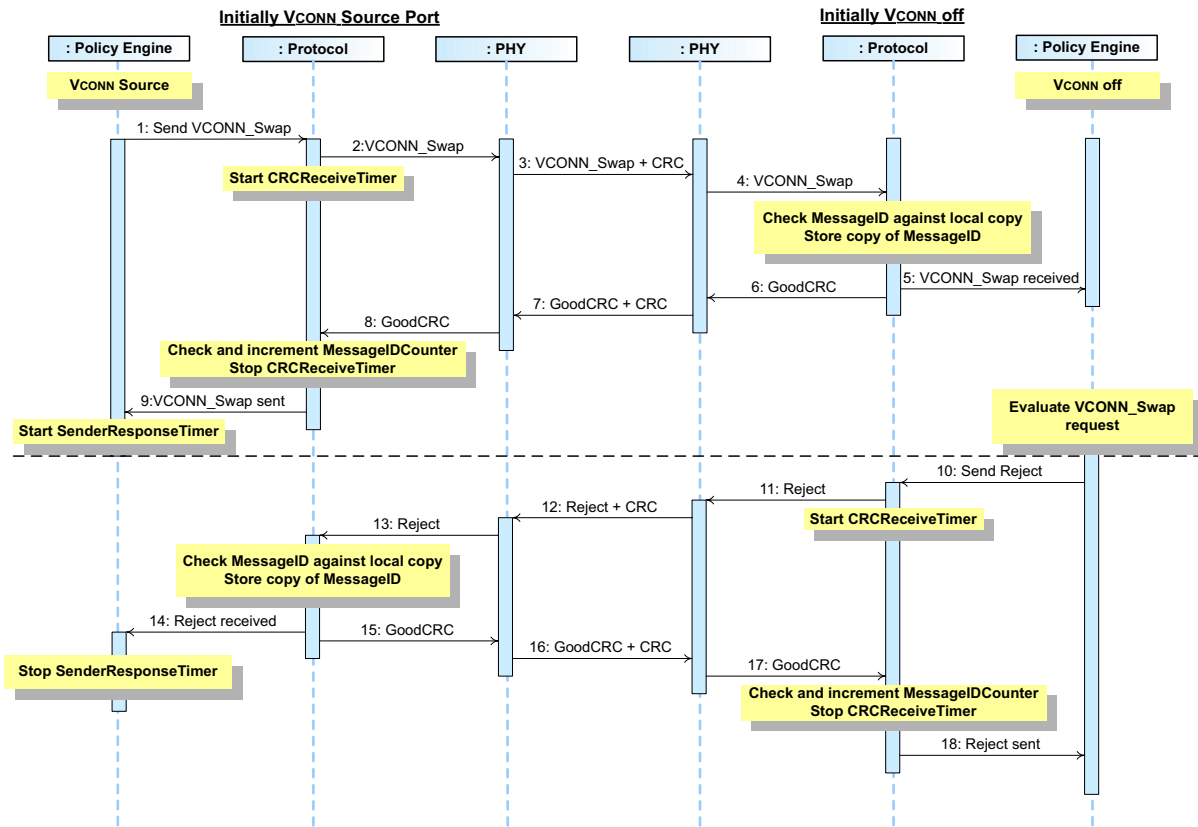
Step	Initially VCONN Source	Initially VCONN off
1	The VCONN Source's Policy Engine directs the Protocol Layer to send a VCONN_Swap Message.	VCONN is off.
2	Protocol Layer creates the VCONN_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the VCONN_Swap Message. Starts CRCReceiveTimer .	PHY Layer receives the VCONN_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the VCONN_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received VCONN_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the VCONN_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the VCONN_Swap Message sent by the Source and decides that it is able and willing to do the VCONN Swap. It tells the Protocol Layer to form an Accept Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Accept Message and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the Accept Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Accept Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer and starts the VCONNOnTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
16	PHY Layer appends a CRC and sends the GoodCRC Message.	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.79 Steps for Source to Sink VCONN Source Swap

Step	Initially VCONN Source	Initially VCONN off
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept</i> Message was successfully sent. The <i>Policy Engine</i> asks the <i>DPM</i> to turn on <i>VCONN</i> .
19		The <i>DPM</i> informs the <i>Policy Engine</i> that its power supply is supplying <i>VCONN</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a <i>PS_RDY</i> Message to tell the <i>Source</i> it can turn off <i>VCONN</i> .
20		<i>Protocol Layer</i> creates the <i>PS_RDY</i> Message and passes to <i>PHY Layer</i> .
21	<i>PHY Layer</i> receives the <i>PS_RDY</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>PS_RDY</i> Message. Starts <i>CRCReceiveTimer</i> .
22	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>PS_RDY</i> Message to the <i>Protocol Layer</i> .	
23	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>PS_RDY</i> Message information to the <i>Policy Engine</i> that consumes it.	
24	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	
25	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message. The <i>Policy Engine</i> stops the <i>VCONNOnTimer</i> , and tells the power supply to stop sourcing <i>VCONN</i> .	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
26		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
27	<i>VCONN</i> is off.	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>PS_RDY</i> Message was successfully sent.
The Port Partners have swapped <i>VCONN</i> Source role.		

Figure 8.53, "Rejected VCONN Source Swap, initiated by VCONN Source" shows an example sequence where the VCONN Source and tells its Port Partner to supply VCONN and is rejected. During the process the Port Partners, keep their Power Role as Source or Sink, maintain their operation as either a Source or a Sink (power remains constant) and don't exchange the VCONN Source role.

Figure 8.53 Rejected VCONN Source Swap, initiated by VCONN Source



[Table 8.80, "Steps for Rejected VCONN Source Swap"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.53, "Rejected VCONN Source Swap, initiated by VCONN Source"](#) above.

Table 8.80 Steps for Rejected VCONN Source Swap

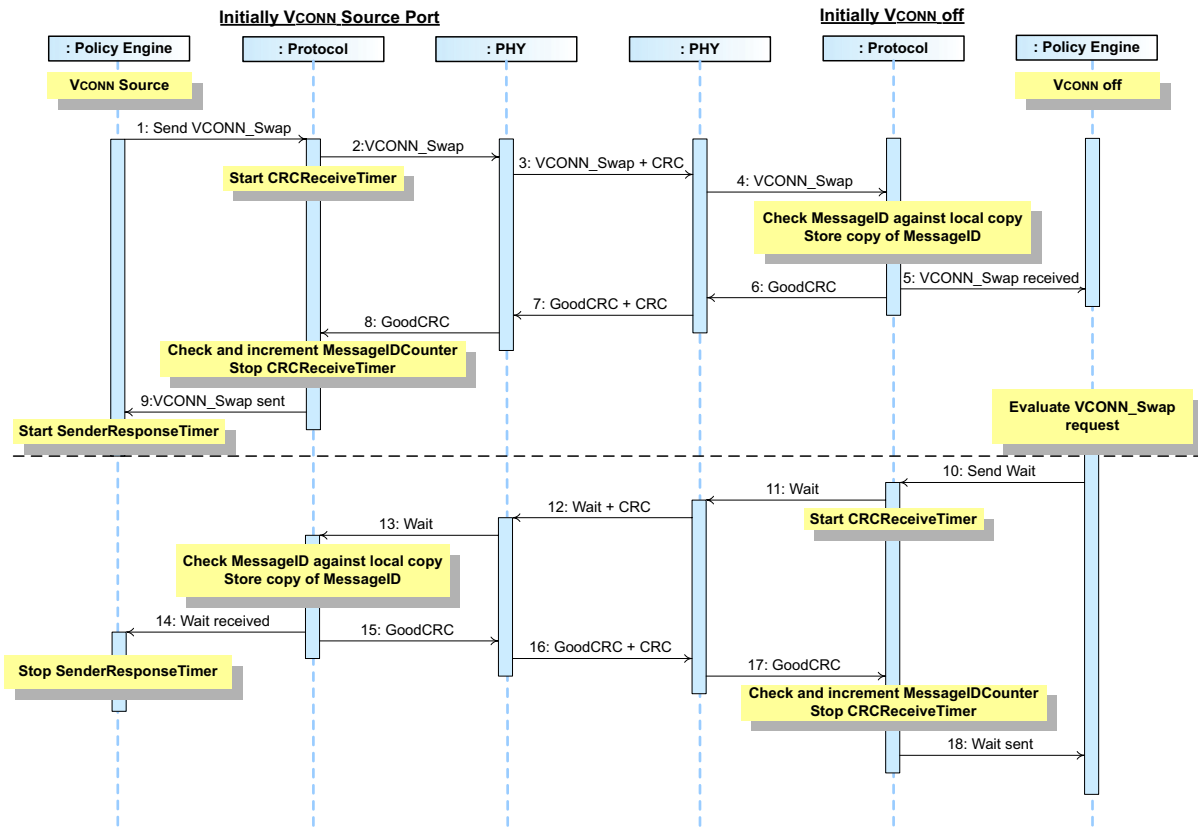
Step	Initially VCONN Source	Initially VCONN off
1	The VCONN Source's Policy Engine directs the Protocol Layer to send a VCONN_Swap Message.	VCONN is off.
2	Protocol Layer creates the VCONN_Swap Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the VCONN_Swap Message. Starts CRCReceiveTimer .	PHY Layer receives the VCONN_Swap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the VCONN_Swap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received VCONN_Swap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the VCONN_Swap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine evaluates the VCONN_Swap Message sent by the Source and decides that it is unable and unwilling to do the VCONN Swap. It tells the Protocol Layer to form an Reject Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Reject Message and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the Reject Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Reject Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer and starts the VCONNOnTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
16	PHY Layer appends a CRC and sends the GoodCRC Message.	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.80 Steps for Rejected VCONN Source Swap

Step	Initially VCONN Source	Initially VCONN off
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject</i> Message was successfully sent

Figure 8.54, "*VCONN Source Swap with Wait, initiated by VCONN Source*" shows an example sequence where the VCONN Source and tells its Port Partner to supply VCONN and is told to wait. During the process the Port Partners, keep their Power Role as Source or Sink, maintain their operation as either a Source or a Sink (power remains constant) and don't exchange the VCONN Source role.

Figure 8.54 VCONN Source Swap with Wait, initiated by VCONN Source



[Table 8.81, "Steps for VCONN Source Swap with Wait"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.54, "VCONN Source Swap with Wait, initiated by VCONN Source"](#) above.

Table 8.81 Steps for VCONN Source Swap with Wait

Step	Initially VCONN Source	Initially VCONN off
1	The <i>VCONN Source's Policy Engine</i> directs the <i>Protocol Layer</i> to send a VCONN_Swap Message.	<i>VCONN</i> is off.
2	<i>Protocol Layer</i> creates the VCONN_Swap Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the VCONN_Swap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the VCONN_Swap Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the VCONN_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received VCONN_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the VCONN_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the VCONN_Swap Message sent by the <i>Source</i> and decides that it is able and willing to do the <i>VCONN Swap</i> but not at this time. It tells the <i>Protocol Layer</i> to form an Wait Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Wait Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Wait Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Wait Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer and starts the VCONNOnTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.

Table 8.81 Steps for VCONN Source Swap with Wait

Step	Initially VCONN Source	Initially VCONN off
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Wait Message was successfully sent</i>

8.3.2.10.2

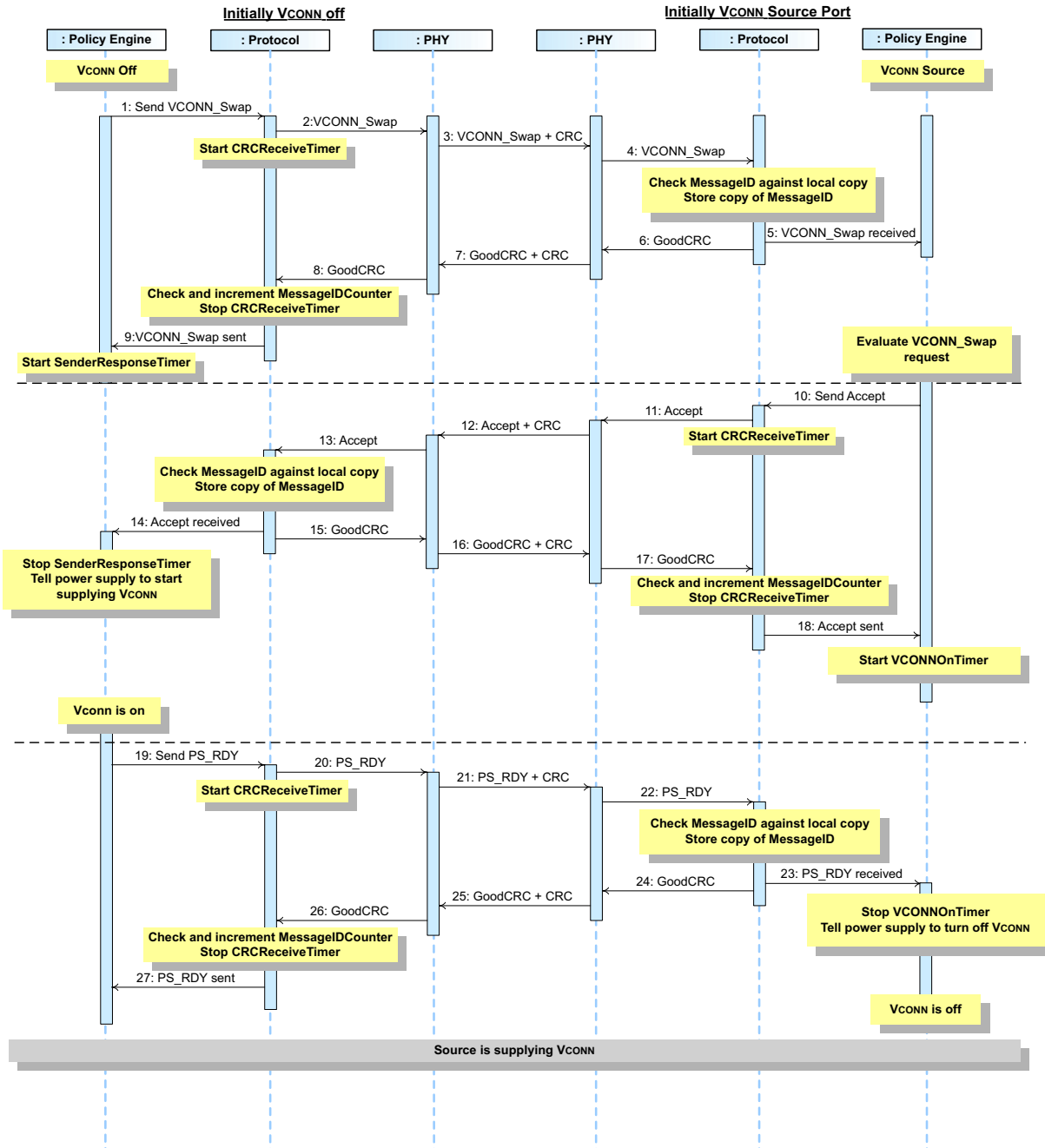
VCONN Source Swap, initiated by non-VCONN Source

8.3.2.10.2.1

VCONN Source Swap, initiated by non-VCONN Source (Accept)

Figure 8.55, "VCONN Source Swap, initiated by non-VCONN Source" shows an example where the *Port* which is not initially supplying *VCONN* and requests a *VCONN Swap*. During the process the *Port Partners*, keep their *Power Role* as *Source* or *Sink*, maintain their operation as either a *Source* or a *Sink* (power remains constant) but exchange the *VCONN Source*.

Figure 8.55 VCONN Source Swap, initiated by non-VCONN Source



[Table 8.82, "Steps for VCONN Source Swap, Initiated by non-VCONN Source"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.55, "VCONN Source Swap, initiated by non-VCONN Source"](#) above.

Table 8.82 Steps for VCONN Source Swap, Initiated by non-VCONN Source

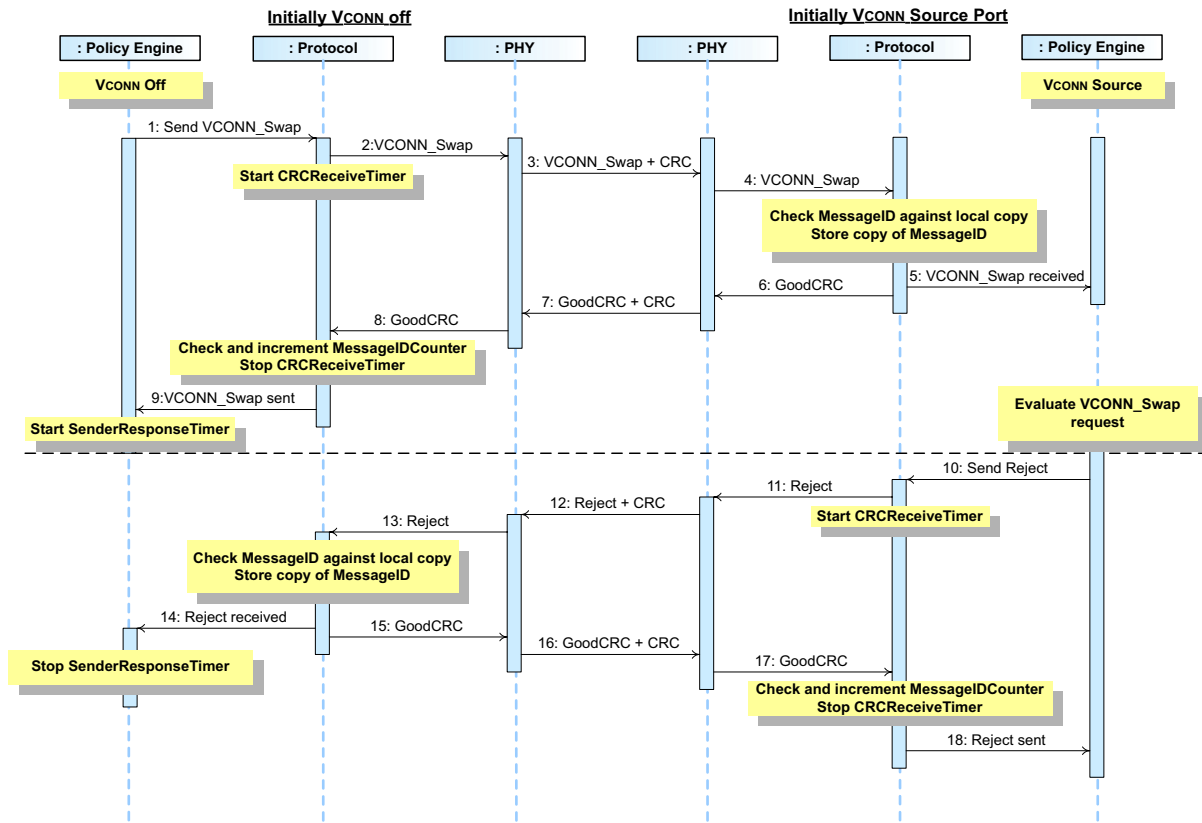
Step	Initially VCONN off	Initially VCONN Source
1	The <i>Source</i> starts with <i>VCONN</i> off. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a VCONN_Swap Message .	The <i>Sink</i> starts as the <i>VCONN Source</i> .
2	<i>Protocol Layer</i> creates the VCONN_Swap Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the VCONN_Swap Message . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the VCONN_Swap Message and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the VCONN_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received VCONN_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the VCONN_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the VCONN_Swap Message sent by the <i>Source</i> and decides that it is able and willing to do the <i>VCONN Swap</i> . It tells the <i>Protocol Layer</i> to form an Accept Message .
11		<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Accept Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Accept Message . Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer . The <i>Policy Engine</i> tells the <i>DPM</i> to turn on <i>VCONN</i> .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message .	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

Table 8.82 Steps for VCONN Source Swap, Initiated by non-VCONN Source

Step	Initially VCONN off	Initially VCONN Source
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Accept Message was successfully sent. The Policy Engine starts the VCONNOnTimer.</i>
19	<i>The DPM tells the Policy Engine that its power supply is supplying VCONN. The Policy Engine directs the Protocol Layer to generate a PS_RDY Message to tell the Sink it can turn off VCONN.</i>	
20	<i>Protocol Layer creates the PS_RDY Message and passes to PHY Layer.</i>	
21	<i>PHY Layer appends a CRC and sends the PS_RDY Message. Starts CRCReceiveTimer.</i>	<i>PHY Layer receives the PS_RDY Message and compares the CRC it calculated with the one sent to verify the Message.</i>
22		<i>PHY Layer removes the CRC and forwards the PS_RDY Message to the Protocol Layer.</i>
23		<i>Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received PS_RDY Message information to the Policy Engine that consumes it.</i>
24		<i>Protocol Layer generates a GoodCRC Message and passes it PHY Layer.</i>
25	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>	<i>PHY Layer appends a CRC and sends the GoodCRC Message. The Policy Engine stops the VCONNOnTimer, and tells the power supply to stop sourcing VCONN.</i>
26	<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>	
27	<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the PS_RDY Message was successfully sent.</i>	<i>VCONN is off.</i>
The Port Partners have swapped VCONN Source role.		

Figure 8.56, "Rejected VCONN Source Swap, initiated by non-VCONN Source" shows an example where the Port which is not initially supplying VCONN and requests a VCONN Swap which is rejected. During the process the Port Partners, keep their Power Role as Source or Sink, maintain their operation as either a Source or a Sink (power remains constant) and don't exchange the VCONN Source.

Figure 8.56 Rejected VCONN Source Swap, initiated by non-VCONN Source



[Table 8.83, "Steps for Rejected VCONN Source Swap, Initiated by non-VCONN Source"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.56, "Rejected VCONN Source Swap, initiated by non-VCONN Source"](#) above.

Table 8.83 Steps for Rejected VCONN Source Swap, Initiated by non-VCONN Source

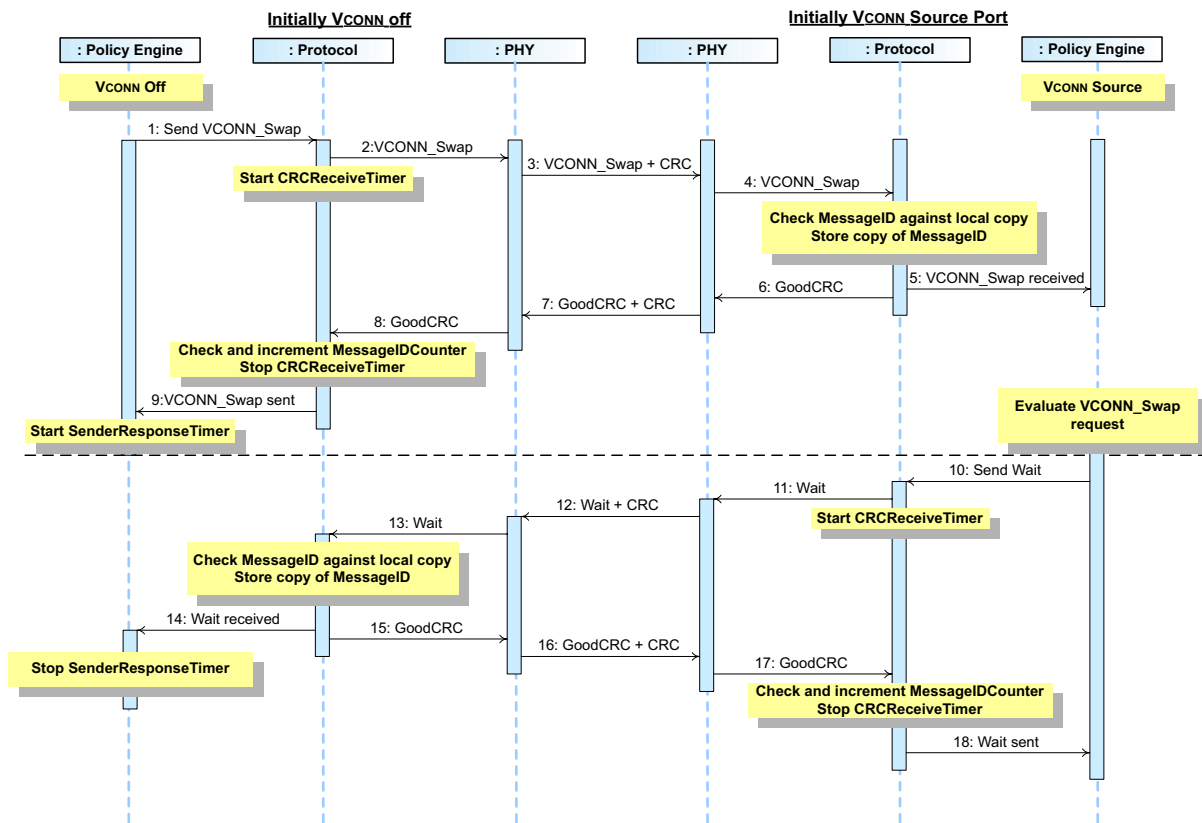
Step	Initially VCONN off	Initially VCONN Source
1	The <i>Source</i> starts with <i>VCONN</i> off. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>VCONN_Swap</i> Message.	The <i>Sink</i> starts as the <i>VCONN Source</i> .
2	<i>Protocol Layer</i> creates the <i>VCONN_Swap</i> Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>VCONN_Swap</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>VCONN_Swap</i> Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>VCONN_Swap</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>VCONN_Swap</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>VCONN_Swap</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> evaluates the <i>VCONN_Swap</i> Message sent by the <i>Source</i> and decides that it is unable and unwilling to do the <i>VCONN Swap</i> . It tells the <i>Protocol Layer</i> to form a <i>Reject</i> Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Reject</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Reject</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Reject</i> Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the <i>SenderResponseTimer</i> . The <i>Policy Engine</i> tells the <i>DPM</i> to turn on <i>VCONN</i> .	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	

Table 8.83 Steps for Rejected VCONN Source Swap, Initiated by non-VCONN Source

Step	Initially VCONN off	Initially VCONN Source
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject</i> Message was successfully sent.

Figure 8.57, "VCONN Source Swap with Wait" shows an example where the *Port* requests a VCONN Swap which is delayed with a wait. During the process the *Port Partners*, keep their *Power Role* as *Source* or *Sink*, maintain their operation as either a *Source* or a *Sink* (power remains constant) and don't exchange the VCONN Source.

Figure 8.57 VCONN Source Swap with Wait



[Table 8.84, "Steps for VCONN Source Swap with Wait"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.57, "VCONN Source Swap with Wait"](#) above.

Table 8.84 Steps for VCONN Source Swap with Wait

Step	Initially VCONN off	Initially VCONN Source
1	The <i>Source</i> starts with <i>VCONN</i> off. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a VCONN_Swap Message.	The <i>Sink</i> starts as the <i>VCONN Source</i> .
2	<i>Protocol Layer</i> creates the VCONN_Swap Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the VCONN_Swap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the VCONN_Swap Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the VCONN_Swap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received VCONN_Swap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the VCONN_Swap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> evaluates the VCONN_Swap Message sent by the <i>Source</i> and decides that it is able and willing to do the <i>VCONN Swap</i> but not at this time. It tells the <i>Protocol Layer</i> to form a Wait Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Wait Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Wait Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Wait Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer . The <i>Policy Engine</i> tells the <i>DPM</i> to turn on <i>VCONN</i> .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.

Table 8.84 Steps for VCONN Source Swap with Wait

Step	Initially VCONN off	Initially VCONN Source
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent.

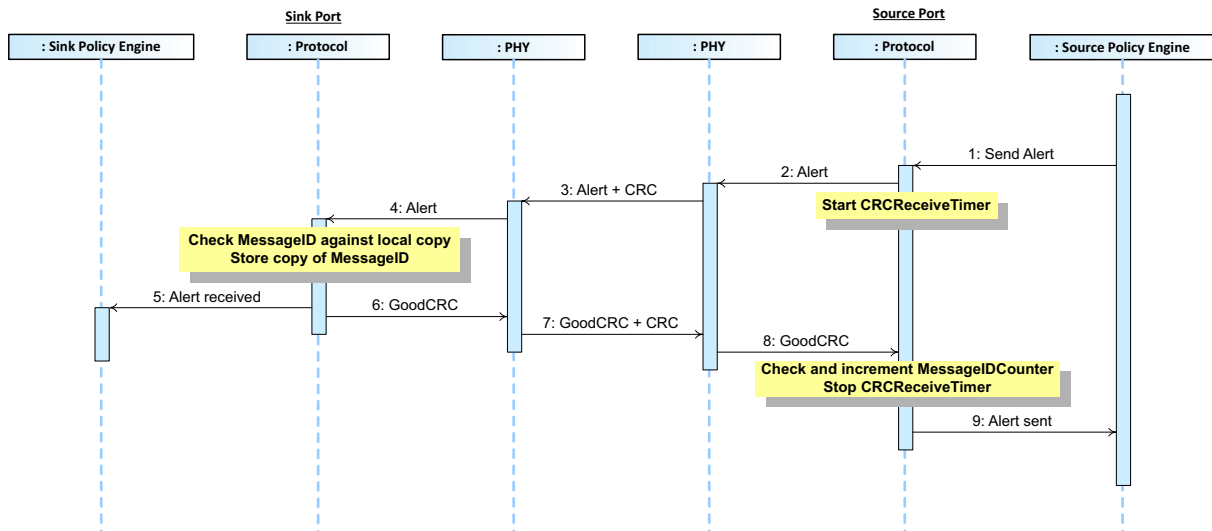
8.3.2.11 Additional Capabilities, Status and Information

8.3.2.11.1 Alert

8.3.2.11.1.1 Source sends Alert to a Sink

Figure 8.58, "Source Alert to Sink" shows an example sequence between a *Source* and a *Sink* where the *Source* alerts the *Sink* that there has been a status change. This AMS will be followed by getting the *Source* status to determine further details of the alert (see [Section 8.3.2.11.2, "Status"](#)).

Figure 8.58 Source Alert to Sink



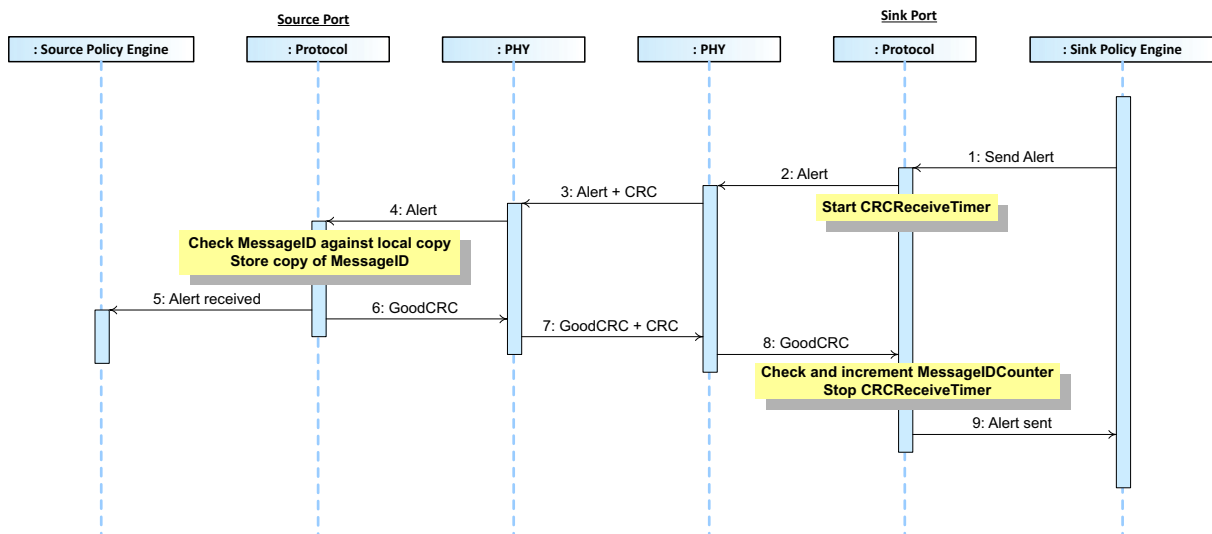
[Table 8.85, "Steps for Source Alert to Sink"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.58, "Source Alert to Sink"](#) above.

Table 8.85 Steps for Source Alert to Sink

Step	Sink	Source
1		The <i>DPM</i> indicates a <i>Source</i> alert condition. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Alert Message .
2		<i>Protocol Layer</i> creates the Alert Message and passes to <i>PHY Layer</i> .
3	<i>PHY Layer</i> receives the Alert Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Alert Message . Starts CRCReceiveTimer .
4	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Alert Message to the <i>Policy Engine</i> that consumes it.	
5	The <i>Policy Engine</i> informs the <i>DPM</i> .	
6	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
7	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message .	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
8		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
9		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Alert Message was successfully sent.

Figure 8.59, "Sink Alert to Source" shows an example sequence between a *Source* and a *Sink* where the *Sink* alerts the *Source* that there has been a status change. This AMS will be followed by getting the *Sink* status to determine further details of the alert (see Section 8.3.2.11.2, "Status").

Figure 8.59 Sink Alert to Source



[Table 8.86, "Steps for Sink Alert to Source"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.59, "Sink Alert to Source"](#) above.

Table 8.86 Steps for Sink Alert to Source

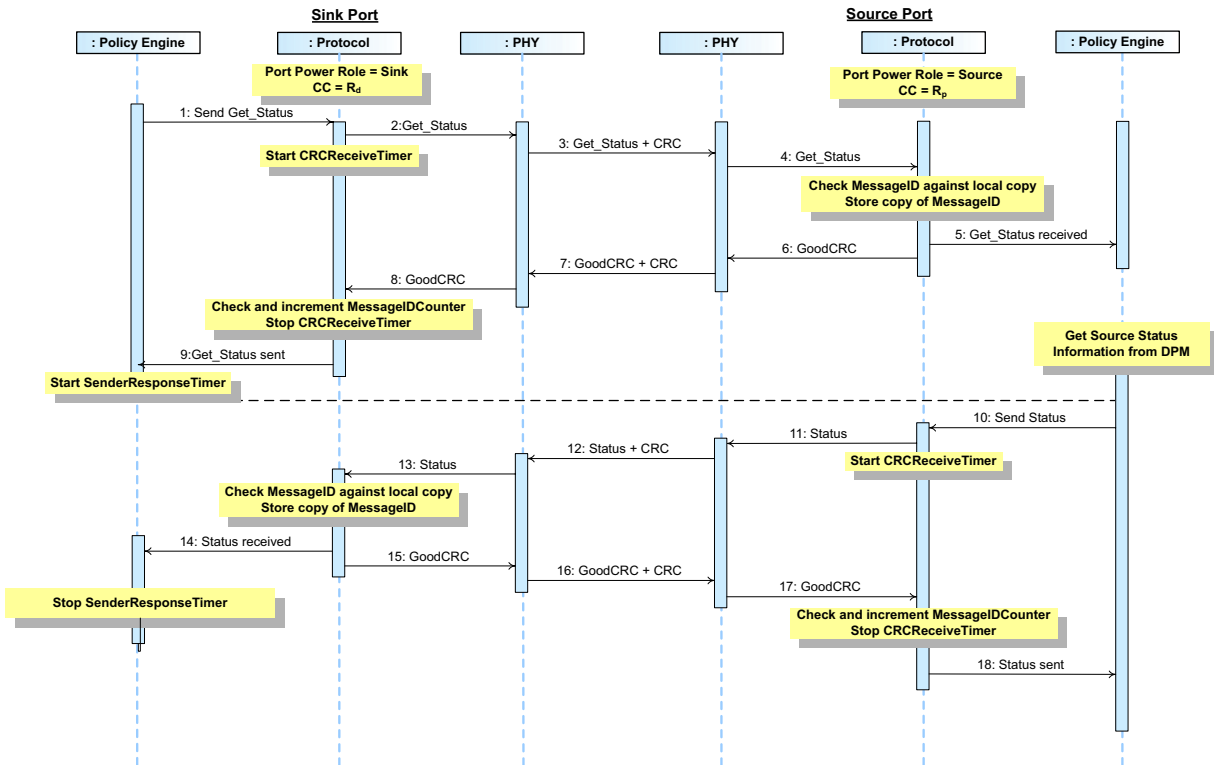
Step	Source	Sink
1		The <i>DPM</i> indicates a <i>Sink</i> alert condition. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Alert Message .
2		<i>Protocol Layer</i> creates the Alert Message and passes to <i>PHY Layer</i> .
3	<i>PHY Layer</i> receives the Alert Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Alert Message . Starts CRCReceiveTimer .
4	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Alert Message to the <i>Policy Engine</i> that consumes it.	
5	The <i>Policy Engine</i> informs the <i>DPM</i> .	
6	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
7	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message .	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
8		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
9		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Alert Message was successfully sent.

8.3.2.11.2 Status

8.3.2.11.2.1 Sink Gets Source Status

Figure 8.60, "Sink Gets Source Status" shows an example sequence between a *Source* and a *Sink* where, after the *Sink* has received an alert (see Section 8.3.2.11.2, "Status") that there has been a status change, the *Sink* gets more details on the change.

Figure 8.60 Sink Gets Source Status



[Table 8.87, "Steps for a Sink getting Source Status Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.60, "Sink Gets Source Status"](#) above.

Table 8.87 Steps for a Sink getting Source Status Sequence

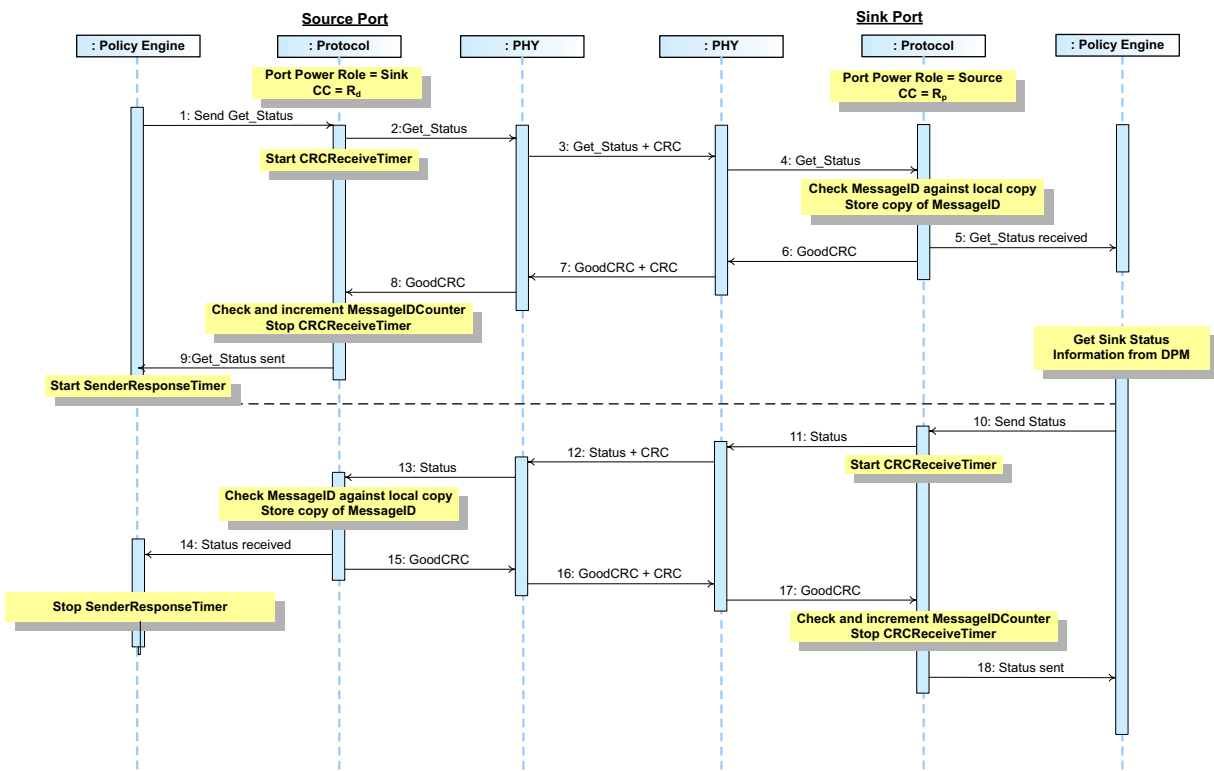
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Status Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Status Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Status Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Status Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Status Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Status Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source status which is provided. The Policy Engine tells the Protocol Layer to form a Status Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Status Message.	PHY Layer appends a CRC and sends the Status Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Status Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.87 Steps for a Sink getting Source Status Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Status Message</i> was successfully sent.
The <i>Source</i> has informed the <i>Sink</i> of its present status.		

Figure 8.61, "Source Gets Sink Status" shows an example sequence between a *Source* and a *Sink* where, after the *Source* has received an alert (see Section 8.3.2.11.2, "Status") that there has been a status change, the *Source* gets more details on the change.

Figure 8.61 Source Gets Sink Status



[Table 8.88, "Steps for a Source getting Sink Status Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.61, "Source Gets Sink Status"](#) above.

Table 8.88 Steps for a Source getting Sink Status Sequence

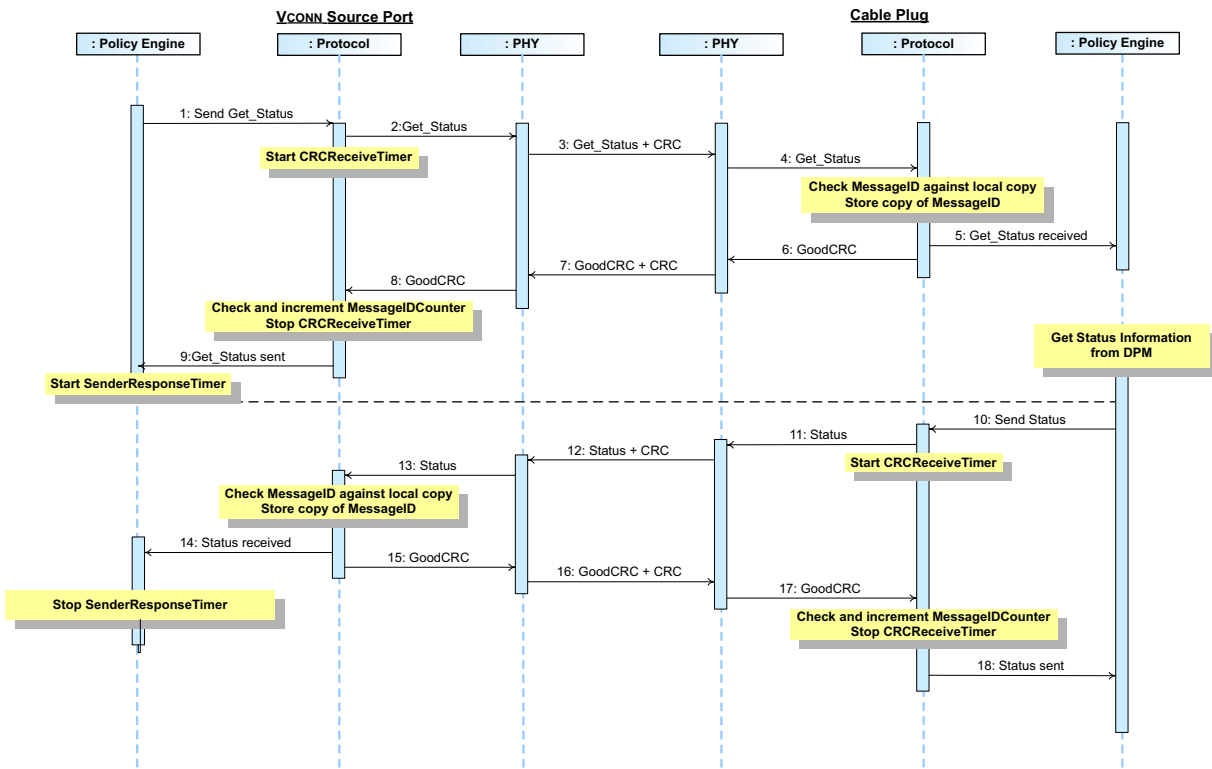
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Status Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Status Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Status Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Status Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Status Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Status Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the present <i>Source</i> status which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Status Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Status Message.	<i>PHY Layer</i> appends a CRC and sends the Status Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Status Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.88 Steps for a Source getting Sink Status Sequence

Step	Source Port	Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Status Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of its present status.		

Figure 8.62, "*VCONN Source Gets Cable Plug Status*" shows an example sequence between a *VCONN Source* and a *Cable Plug* where, after the *VCONN Source* has received an alert (see [Section 8.3.2.11.2, "Status"](#)) that there has been a status change, the *VCONN Source* gets more details on the change.

Figure 8.62 VCONN Source Gets Cable Plug Status



[Table 8.89, "Steps for a VCONN Source getting Cable Plug Status Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.62, "VCONN Source Gets Cable Plug Status"](#) above.

Table 8.89 Steps for a VCONN Source getting Cable Plug Status Sequence

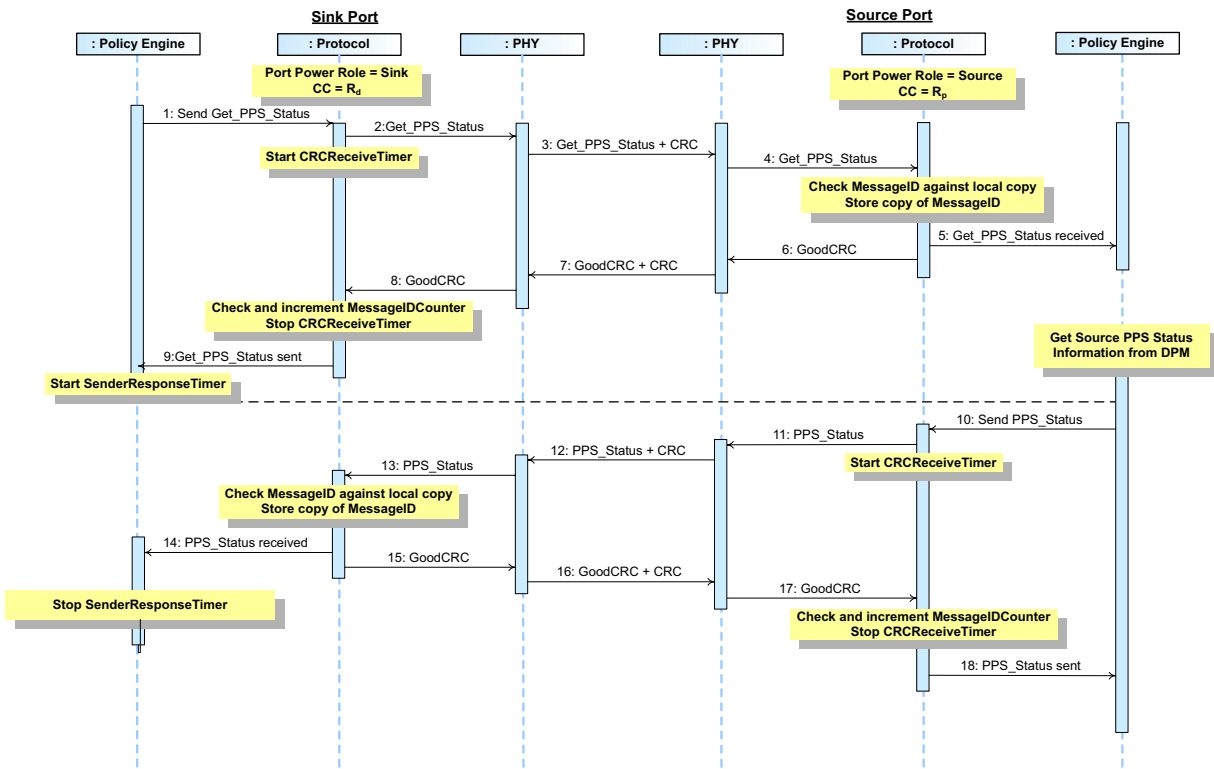
Step	VCONN Source Port	Cable Plug
1	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Status Message.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Status Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Status Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Status Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Status Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Status Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present Source status which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Status Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Status Message.	<i>PHY Layer</i> appends a CRC and sends the Status Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Status Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .

Table 8.89 Steps for a VCONN Source getting Cable Plug Status Sequence

Step	VCONN Source Port	Cable Plug
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Status Message</i> was successfully sent.
The <i>Cable Plug</i> has informed the <i>VCONN Source</i> of its present status.		

Figure 8.63, "Sink Gets Source PPS Status" shows an example sequence between a *Source* and a *Sink* where, after the *Sink* has received an alert (see Section 8.3.2.11.2, "Status") that there has been a PPS status change, the *Sink* gets more details on the change.

Figure 8.63 Sink Gets Source PPS Status



[Table 8.90, "Steps for a Sink getting Source PPS status Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.63, "Sink Gets Source PPS Status"](#) above.

Table 8.90 Steps for a Sink getting Source PPS status Sequence

Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_PPS_Status Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_PPS_Status Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_PPS_Status Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Status Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_PPS_Status Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_PPS_Status Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source status which is provided. The Policy Engine tells the Protocol Layer to form a PPS_Status Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the PPS_Status Message.	PHY Layer appends a CRC and sends the PPS_Status Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received PPS_Status Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.90 Steps for a Sink getting Source PPS status Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>PPS_Status Message</i> was successfully sent.
The <i>Source</i> has informed the <i>Sink</i> of its present PPS status.		

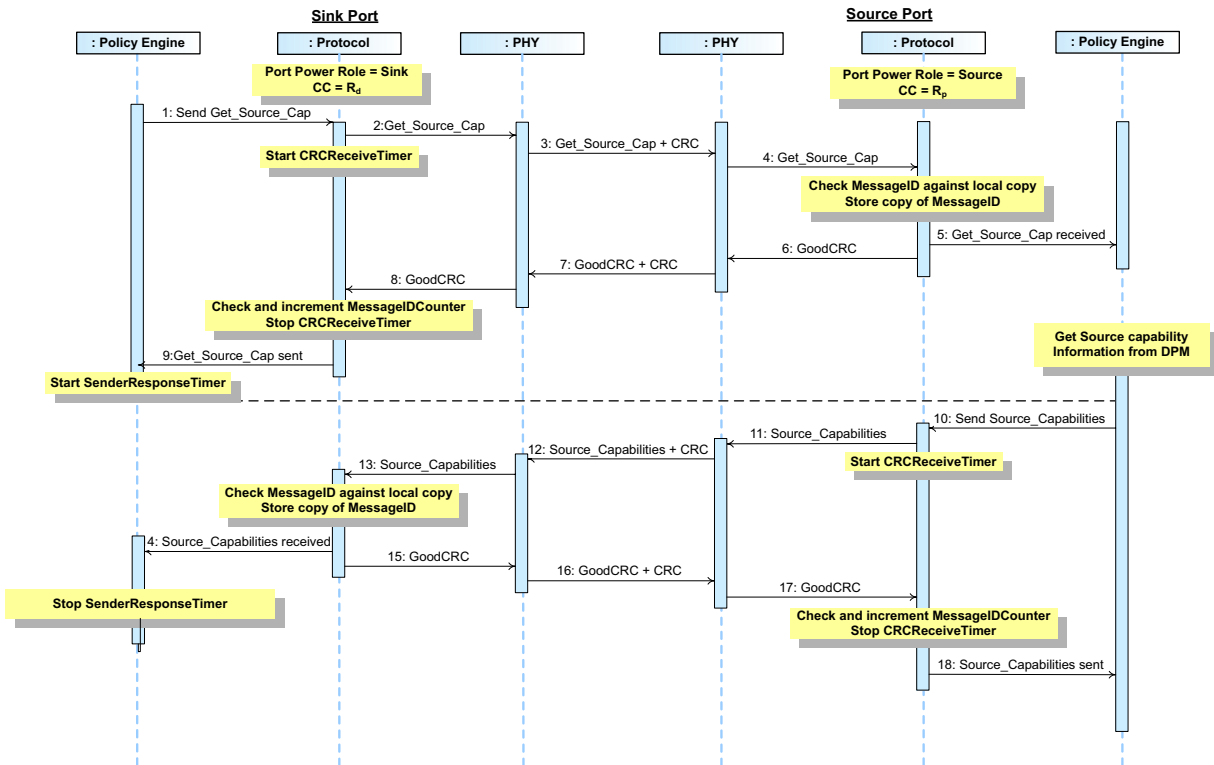
8.3.2.11.3 Source/Sink Capabilities

8.3.2.11.3.1 SPR

8.3.2.11.3.1.1 Sink Gets Source Capabilities

Figure 8.64, "Sink Gets Source's Capabilities" shows an example sequence between a *Source* and a *Sink* when the *Sink* gets the *Source Capabilities*.

Figure 8.64 Sink Gets Source's Capabilities



[Table 8.91, "Steps for a Sink getting Source Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.64, "Sink Gets Source's Capabilities"](#) above.

Table 8.91 Steps for a Sink getting Source Capabilities Sequence

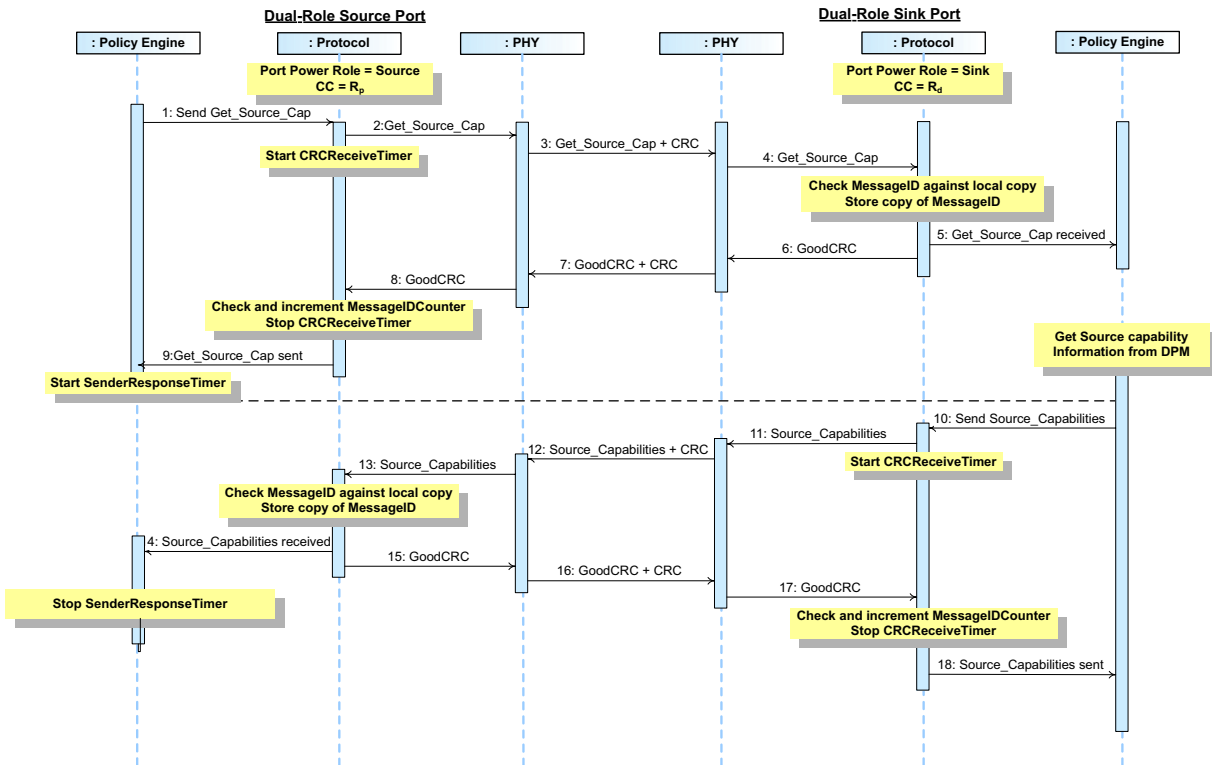
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Source_Cap Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Source_Cap Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Source_Cap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Source_Cap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Source_Cap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Source_Cap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source Capabilities which are provided. The Policy Engine tells the Protocol Layer to form a Source_Capabilities Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Source_Capabilities Message.	PHY Layer appends a CRC and sends the Source_Capabilities Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Source_Capabilities Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.91 Steps for a Sink getting Source Capabilities Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Source_Capabilities Message was successfully sent.</i>
The Source has informed the Sink of its capabilities.		

Figure 8.65, "Dual-Role Source Gets Dual-Role Sink's Capabilities as a Source" shows an example sequence between a Dual-Role Power Source and a Dual-Role Power Sink when the Source gets the Sink Capabilities as a Source.

Figure 8.65 Dual-Role Source Gets Dual-Role Sink's Capabilities as a Source



[Table 8.92, "Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as a Source Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.65, "Dual-Role Source Gets Dual-Role Sink's Capabilities as a Source"](#) above.

Table 8.92 Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as a Source Sequence

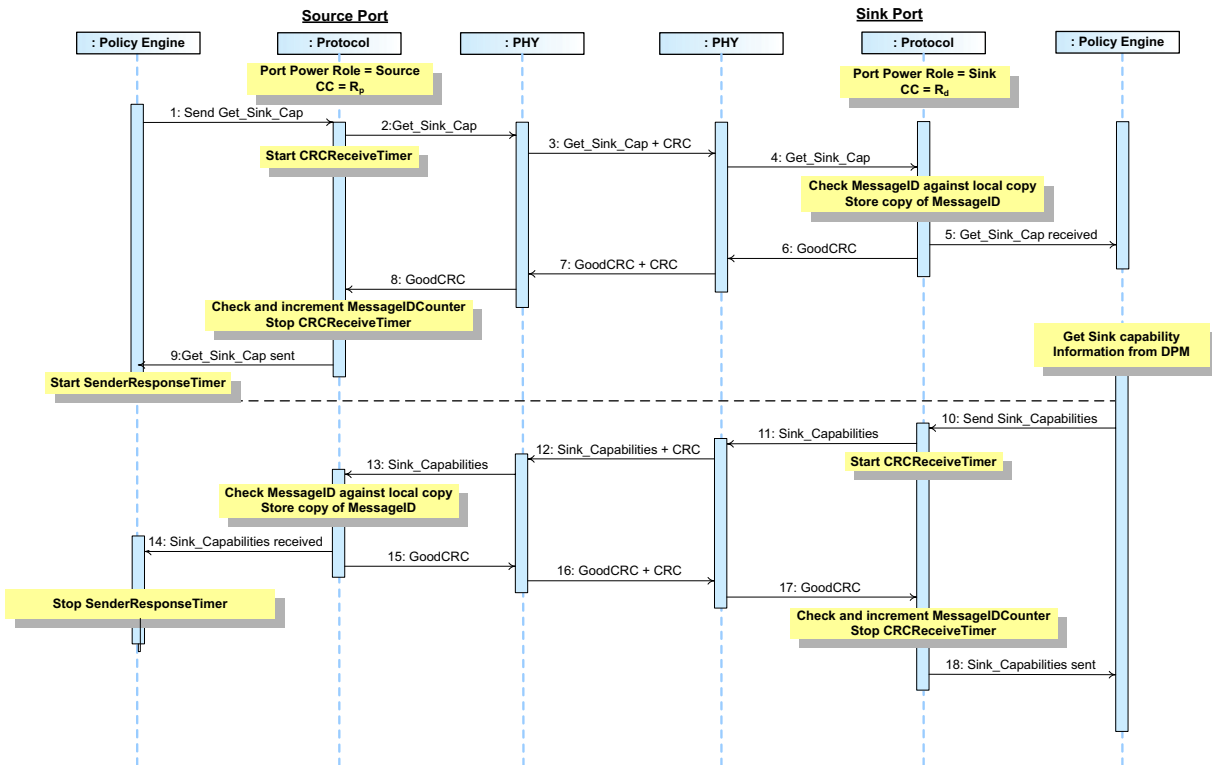
Step	Dual-Role Source Port	Dual-Role Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Source_Cap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Source_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Source_Cap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Source_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Source_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Source_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present <i>Source Capabilities</i> which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Source_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Source_Capabilities Message.	<i>PHY Layer</i> appends a CRC and sends the Source_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Source_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.92 Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as a Source Sequence

Step	Dual-Role Source Port	Dual-Role Sink Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Source_Capabilities Message was successfully sent.</i>
The Dual-Role Power Sink has informed the Dual-Role Power Source of its capabilities.		

Figure 8.66, "Source Gets Sink's Capabilities" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink* Capabilities.

Figure 8.66 Source Gets Sink's Capabilities



[Table 8.93, "Steps for a Source getting Sink Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.66, "Source Gets Sink's Capabilities"](#) above.

Table 8.93 Steps for a Source getting Sink Capabilities Sequence

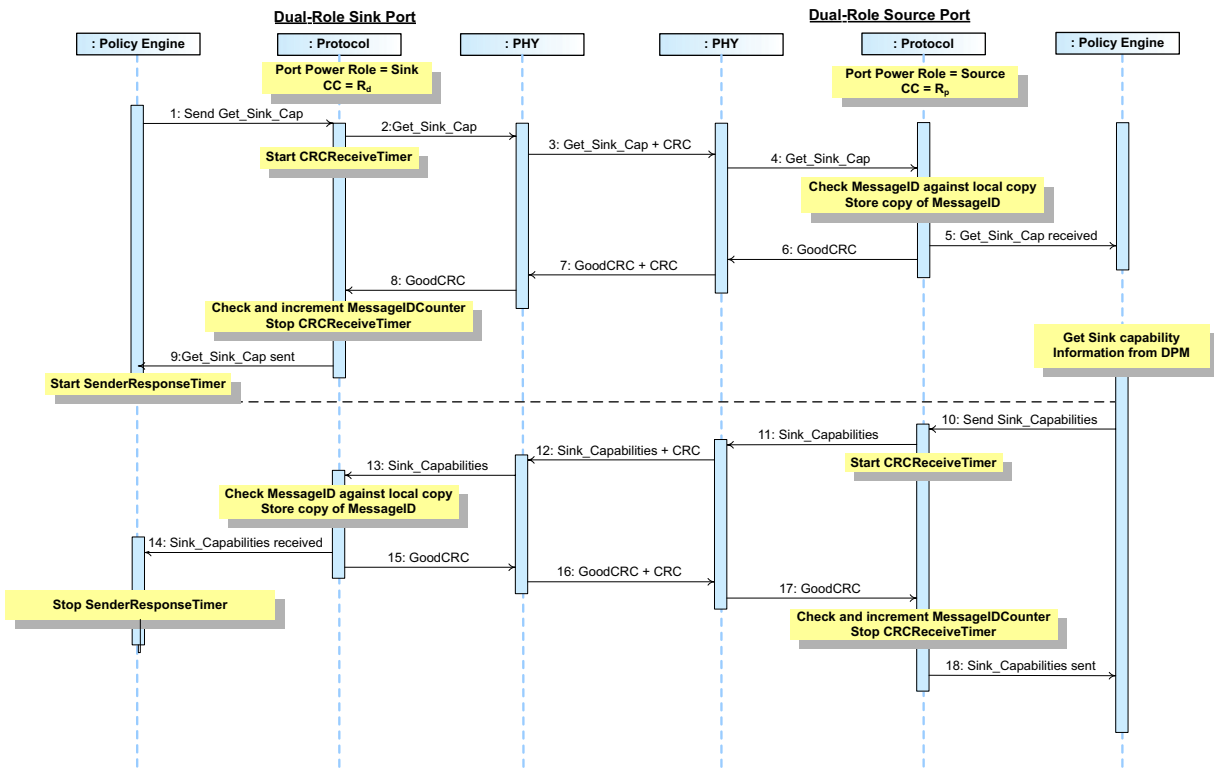
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Sink_Cap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Sink_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Sink_Cap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Sink_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Sink_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Sink_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present Sink Capabilities which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Sink_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Sink_Capabilities Message.	<i>PHY Layer</i> appends a CRC and sends the Sink_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Sink_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.93 Steps for a Source getting Sink Capabilities Sequence

Step	Source Port	Sink Port
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Sink_Capabilities Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of its capabilities.		

Figure 8.67, "Dual-Role Sink Gets Dual-Role Source's Capabilities as a Sink" shows an example sequence between a Dual-Role Power Source and a Dual-Role Power Sink when the Dual-Role Power Sink gets the Dual-Role Power Source Capabilities as a Sink.

Figure 8.67 Dual-Role Sink Gets Dual-Role Source's Capabilities as a Sink



[Table 8.94, "Steps for a Dual-Role Sink getting Dual-Role Source capabilities as a Sink Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.67, "Dual-Role Sink Gets Dual-Role Source's Capabilities as a Sink"](#) above.

Table 8.94 Steps for a Dual-Role Sink getting Dual-Role Source capabilities as a Sink Sequence

Step	Dual-Role Sink Port	Dual-Role Source Port
1	The Port has Port Power Role set to <i>Dual-Role Power Sink</i> with the R_d pull down on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Sink_Cap Message.	The Port has Port Power Role set to <i>Dual-Role Power Source</i> and the R_p pull up on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Sink_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Sink_Cap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Sink_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Sink_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Sink_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present <i>Dual-Role Power Source Capabilities</i> which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Sink_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Sink_Capabilities Message.	<i>PHY Layer</i> appends a CRC and sends the Sink_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Sink_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.94 Steps for a Dual-Role Sink getting Dual-Role Source capabilities as a Sink Sequence

Step	Dual-Role Sink Port	Dual-Role Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Sink_Capabilities Message</i> was successfully sent.
The <i>Dual-Role Power Source</i> has informed the <i>Dual-Role Power Sink</i> of its <i>Capabilities</i> as a <i>Sink</i> .		

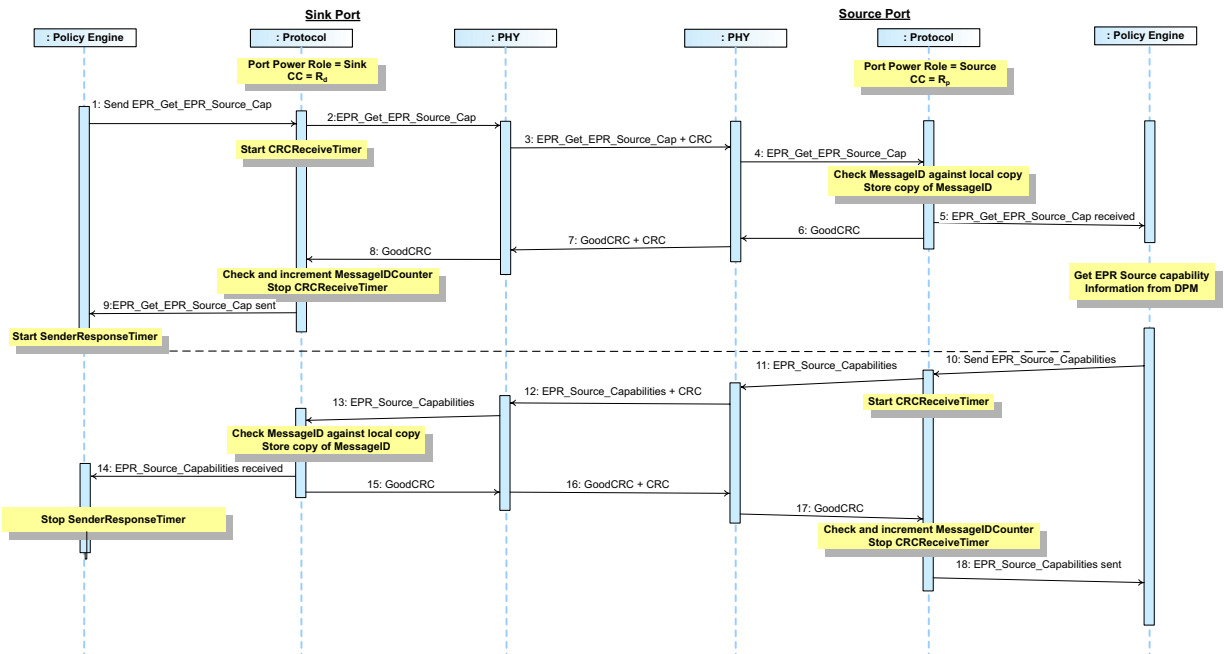
EPR

8.3.2.11.3.2.1

Sink Gets EPR Source Capabilities

Figure 8.68, "Sink Gets Source's EPR Capabilities" shows an example sequence between a *Source* and a *Sink* when the *Sink* gets the *Source's* *EPR Capabilities*.

Figure 8.68 Sink Gets Source's EPR Capabilities



[Table 8.95, "Steps for a Sink getting EPR Source Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.68, "Sink Gets Source's EPR Capabilities"](#) above.

Table 8.95 Steps for a Sink getting EPR Source Capabilities Sequence

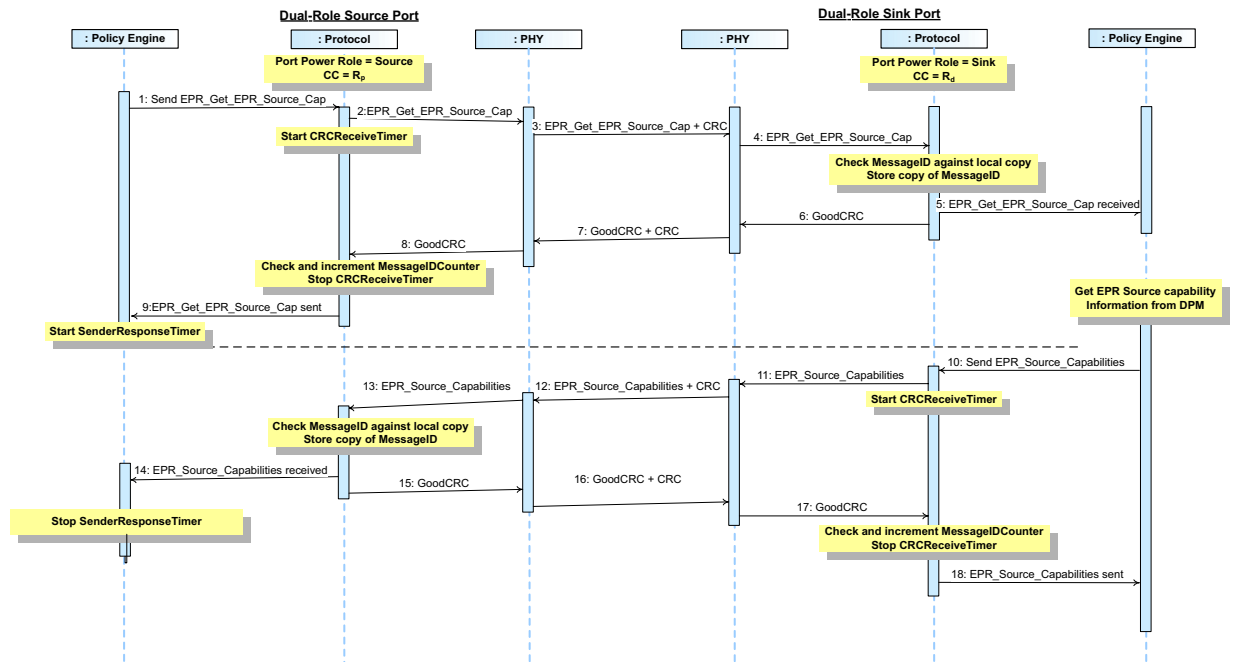
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a EPR_Get_Source_Cap Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the EPR_Get_Source_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Get_Source_Cap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the EPR_Get_Source_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Get_Source_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Get_Source_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present EPR Source Capabilities which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an EPR_Source_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the EPR_Source_Capabilities Message.	<i>PHY Layer</i> appends a CRC and sends the EPR_Source_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Source_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.95 Steps for a Sink getting EPR Source Capabilities Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Capabilities Message</i> was successfully sent.
The <i>Source</i> has informed the <i>Sink</i> of its <i>EPR Capabilities</i> .		

Figure 8.69, "Dual-Role Source Gets Dual-Role Sink's Capabilities as an EPR Source" shows an example sequence between a Dual-Role Power Source and a Dual-Role Power Sink when the Source gets the Sink Capabilities as an EPR Source.

Figure 8.69 Dual-Role Source Gets Dual-Role Sink's Capabilities as an EPR Source



[Table 8.96, "Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as an EPR Source Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.69, "Dual-Role Source Gets Dual-Role Sink's Capabilities as an EPR Source"](#) above.

Table 8.96 Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as an EPR Source Sequence

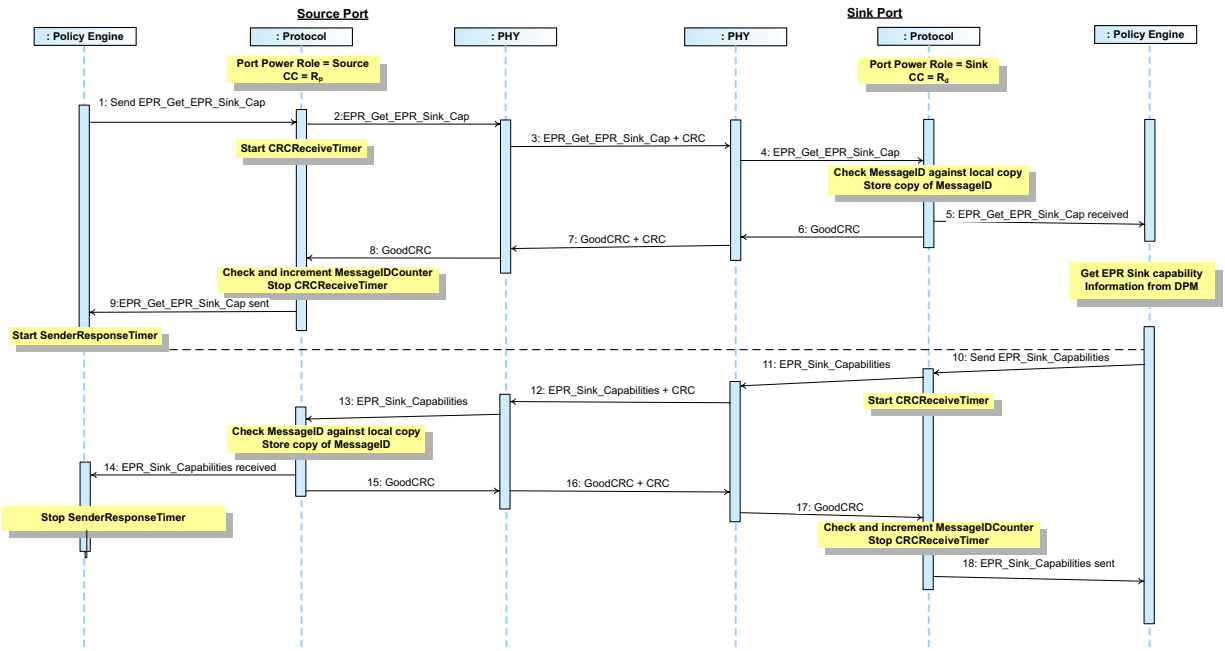
Step	Dual-Role Source Port	Dual-Role Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a EPR_Get_Source_Cap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the EPR_Get_Source_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Get_Source_Cap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the EPR_Get_Source_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Get_Source_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Get_Source_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the present <i>Source Capabilities</i> which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an EPR_Source_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the EPR_Source_Capabilities Message.	<i>PHY Layer</i> appends a CRC and sends the EPR_Source_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Source_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.96 Steps for a Dual-Role Source getting Dual-Role Sink's capabilities as an EPR Source Sequence

Step	Dual-Role Source Port	Dual-Role Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>EPR_Source_Capabilities</i> Message was successfully sent.
The <i>Dual-Role Power Sink</i> has informed the <i>Dual-Role Power Source</i> of its <i>EPR Capabilities</i> .		

Figure 8.70, "Source Gets Sink's EPR Capabilities" shows an example sequence between a Source and a Sink when the Source gets the Sink's EPR Capabilities.

Figure 8.70 Source Gets Sink's EPR Capabilities



[Table 8.97, "Steps for a Source getting Sink EPR Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.70, "Source Gets Sink's EPR Capabilities"](#) above.

Table 8.97 Steps for a Source getting Sink EPR Capabilities Sequence

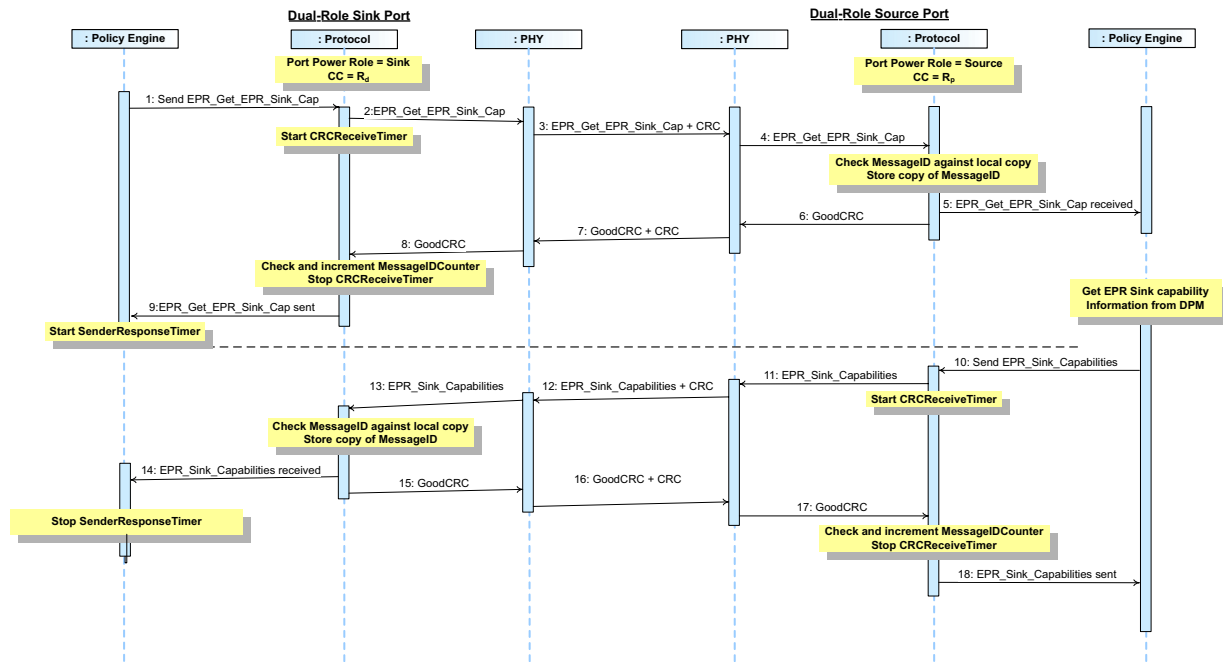
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. Policy Engine directs the Protocol Layer to send a EPR_Get_Sink_Cap Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the EPR_Get_Sink_Cap Message. Starts CRCReceiveTimer .	PHY Layer receives the EPR_Get_Sink_Cap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the EPR_Get_Sink_Cap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received EPR_Get_Sink_Cap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the EPR_Get_Sink_Cap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Sink Capabilities which are provided. The Policy Engine tells the Protocol Layer to form an EPR_Sink_Capabilities Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the EPR_Sink_Capabilities Message.	PHY Layer appends a CRC and sends the EPR_Sink_Capabilities Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received EPR_Sink_Capabilities Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.97 Steps for a Source getting Sink EPR Capabilities Sequence

Step	Source Port	Sink Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the EPR_Sink_Capabilities Message was successfully sent.</i>
The Sink has informed the Source of its EPR Capabilities.		

Figure 8.71, "Dual-Role Sink Gets Dual-Role Source's Capabilities as an EPR Sink" shows an example sequence between a Dual-Role Power Source and a Dual-Role Power Sink when the Dual-Role Power Sink gets the Dual-Role Power Source Capabilities as a Sink.

Figure 8.71 Dual-Role Sink Gets Dual-Role Source's Capabilities as an EPR Sink



[Table 8.98, "Steps for a Dual-Role Sink getting Dual-Role Source Capabilities as an EPR Sink Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.71, "Dual-Role Sink Gets Dual-Role Source's Capabilities as an EPR Sink"](#) above.

Table 8.98 Steps for a Dual-Role Sink getting Dual-Role Source Capabilities as an EPR Sink Sequence

Step	Dual-Role Sink Port	Dual-Role Source Port
1	The Port has Port Power Role set to <i>Dual-Role Power Sink</i> with the R_d pull down on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a EPR_Get_Sink_Cap Message.	The Port has Port Power Role set to <i>Dual-Role Power Source</i> and the R_p pull up on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the EPR_Get_Sink_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the EPR_Get_Sink_Cap Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the EPR_Get_Sink_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Get_Sink_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the EPR_Get_Sink_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present <i>Dual-Role Power Source Capabilities</i> which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an EPR_Sink_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the EPR_Sink_Capabilities Message.	<i>PHY Layer</i> appends a CRC and sends the EPR_Sink_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received EPR_Sink_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.98 Steps for a Dual-Role Sink getting Dual-Role Source Capabilities as an EPR Sink Sequence

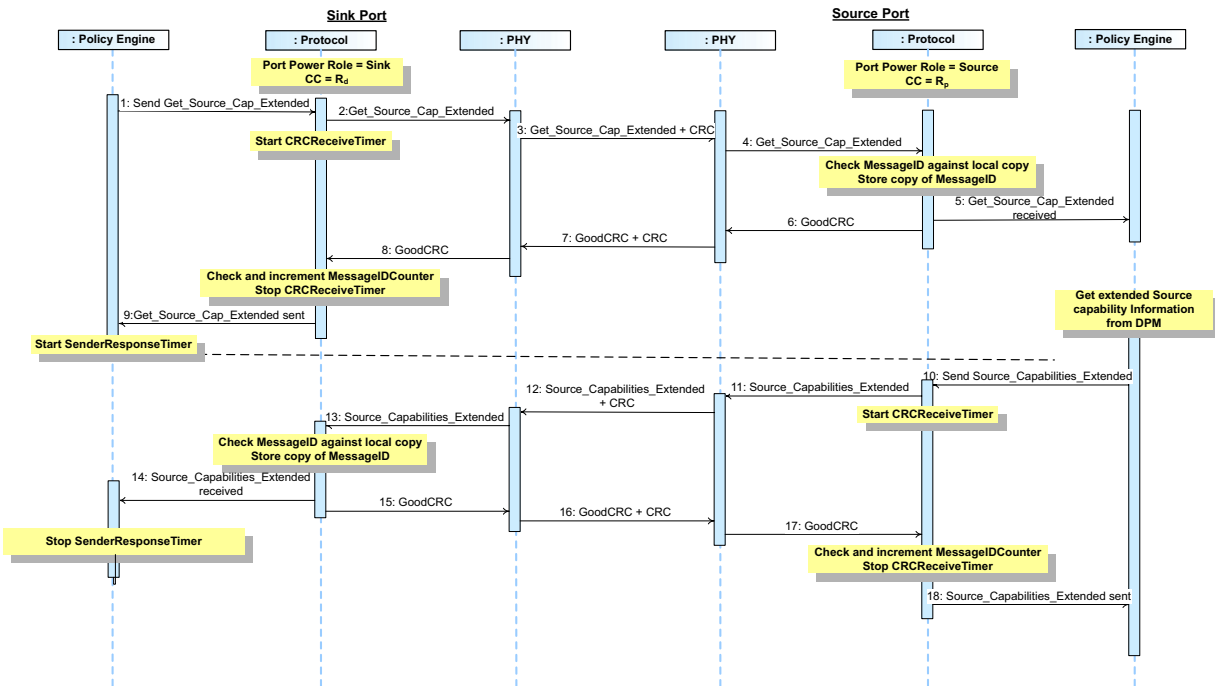
Step	Dual-Role Sink Port	Dual-Role Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the EPR_Sink_Capabilities Message was successfully sent.</i>
The Dual-Role Power Source has informed the Dual-Role Power Sink of its Capabilities as an EPR Sink.		

8.3.2.11.4 Extended Capabilities

8.3.2.11.4.1 Sink Gets Source Extended Capabilities

Figure 8.72, "Sink Gets Source's Extended Capabilities" shows an example sequence between a *Source* and a *Sink* when the *Sink* gets the *Source*'s *Extended Capabilities*.

Figure 8.72 Sink Gets Source's Extended Capabilities



[Table 8.99, "Steps for a Sink getting Source Extended Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.72, "Sink Gets Source's Extended Capabilities"](#) above.

Table 8.99 Steps for a Sink getting Source Extended Capabilities Sequence

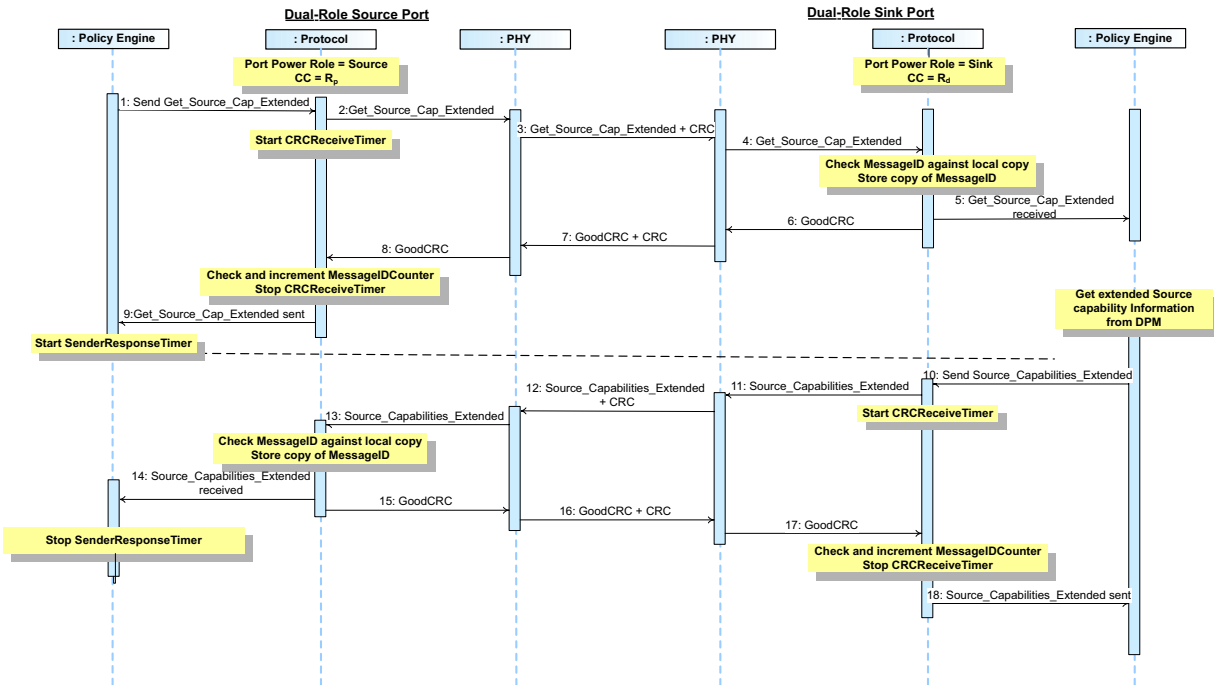
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Source_Cap_Extended Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Source_Cap_Extended Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Source_Cap_Extended Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Source_Cap_Extended Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Source_Cap_Extended Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Source_Cap_Extended Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source Extended Capabilities which are provided. The Policy Engine tells the Protocol Layer to form a Source_Capabilities_Extended Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Source_Capabilities_Extended Message.	PHY Layer appends a CRC and sends the Source_Capabilities_Extended Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Source_Capabilities_Extended Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.99 Steps for a Sink getting Source Extended Capabilities Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Source_Capabilities_Extended Message was successfully sent.</i>
The Source has informed the Sink of its <i>Extended Capabilities</i> .		

Figure 8.73, "*Dual-Role Source Gets Dual-Role Sink's Extended Capabilities*" shows an example sequence between a Source and a Sink when the Dual-Role Power Source gets the Dual-Role Power Sink's Extended Capabilities as a Source.

Figure 8.73 Dual-Role Source Gets Dual-Role Sink's Extended Capabilities



[Table 8.100, "Steps for a Dual-Role Source getting Dual-Role Sink Extended Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.73, "Dual-Role Source Gets Dual-Role Sink's Extended Capabilities"](#) above.

Table 8.100 Steps for a Dual-Role Source getting Dual-Role Sink Extended Capabilities Sequence

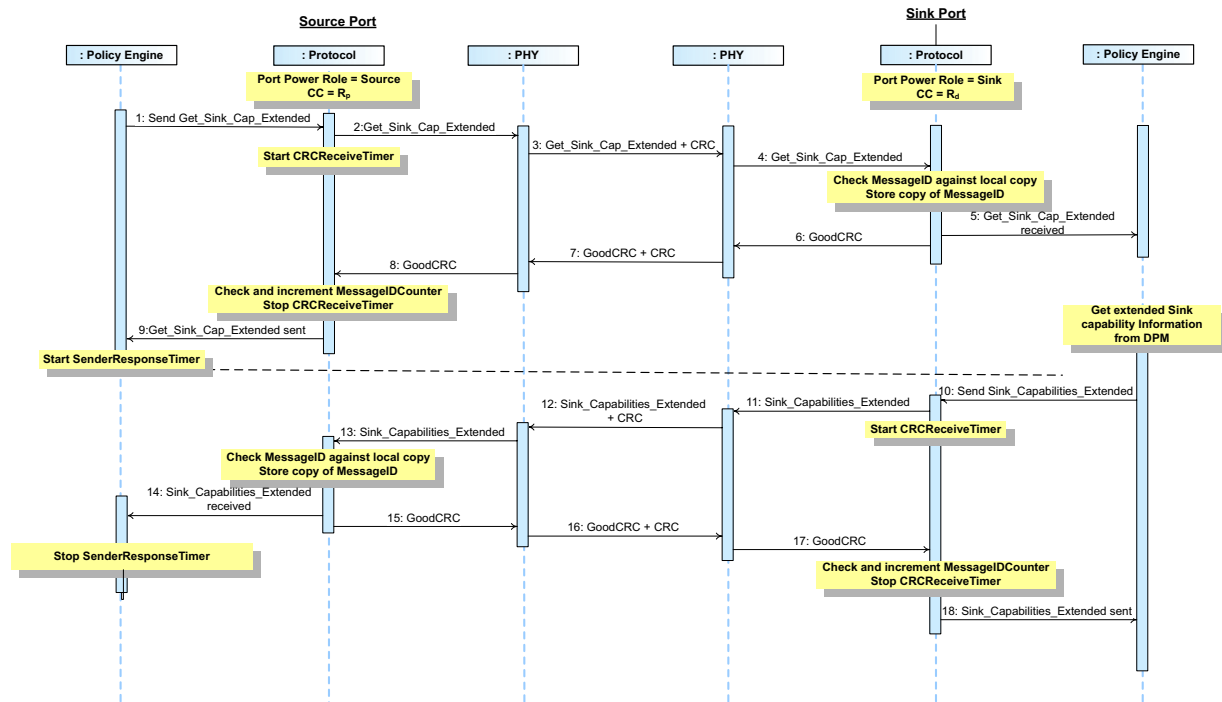
Step	Dual-Role Source Port	Dual-Role Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Source_Cap_Extended Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Source_Cap_Extended Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Source_Cap_Extended Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Source_Cap_Extended Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Source_Cap_Extended Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Source_Cap_Extended Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the Extended Capabilities as a Source which are provided. The Policy Engine tells the Protocol Layer to form a Source_Capabilities_Extended Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Source_Capabilities_Extended Message.	PHY Layer appends a CRC and sends the Source_Capabilities_Extended Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Source_Capabilities_Extended Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	

Table 8.100 Steps for a Dual-Role Source getting Dual-Role Sink Extended Capabilities Sequence

Step	Dual-Role Source Port	Dual-Role Sink Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Source_Capabilities_Extended Message was successfully sent.
The Dual-Role Power Sink has informed the Dual-Role Power Source of its Extended Capabilities as a Source.		

Figure 8.74, "Source Gets Sink's Extended Capabilities" shows an example sequence between a Source and a Sink when the Source gets the Sink's Extended Capabilities.

Figure 8.74 Source Gets Sink's Extended Capabilities



[Table 8.101, "Steps for a Source getting Sink Extended Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.74, "Source Gets Sink's Extended Capabilities"](#) above.

Table 8.101 Steps for a Source getting Sink Extended Capabilities Sequence

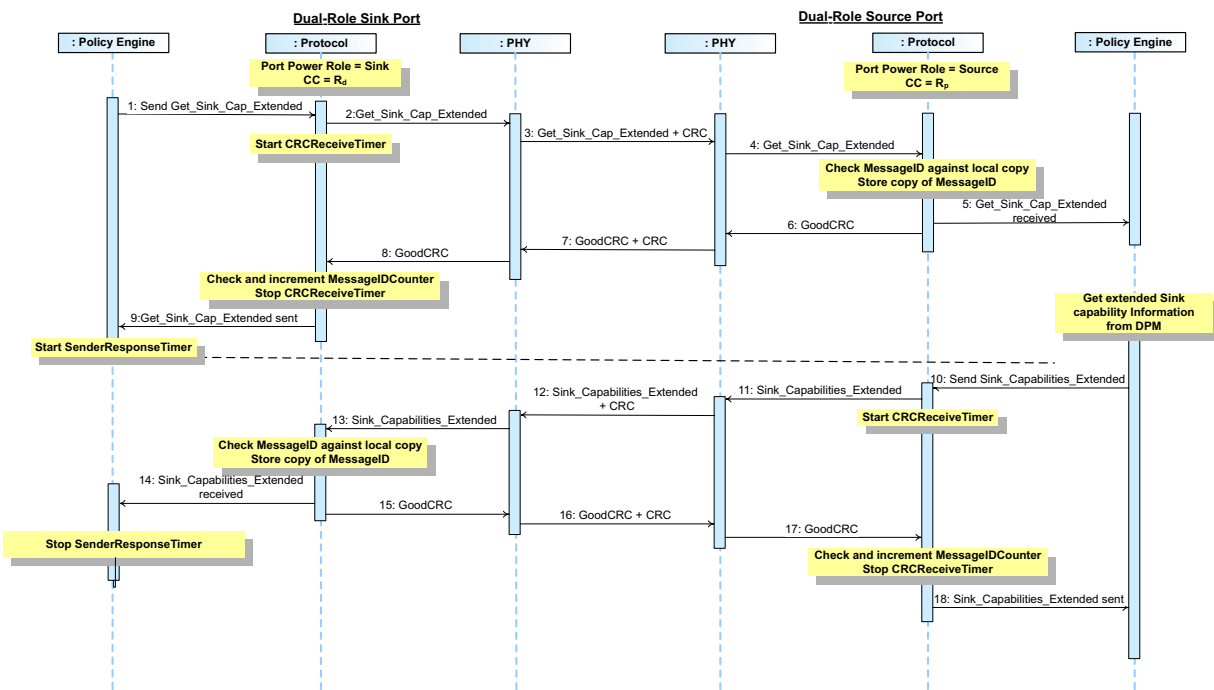
Step	Sink Port	Source Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Sink_Cap_Extended Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Source_Cap_Extended Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Sink_Cap_Extended Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Sink_Cap_Extended Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Sink_Cap_Extended Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Sink_Cap_Extended Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the present Sink Extended Capabilities which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Sink_Capabilities_Extended Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Sink_Capabilities_Extended Message.	<i>PHY Layer</i> appends a CRC and sends the Sink_Capabilities_Extended Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Sink_Capabilities_Extended Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.101 Steps for a Source getting Sink Extended Capabilities Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Sink_Capabilities_Extended Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of its <i>Extended Capabilities</i> .		

Figure 8.75, "Dual-Role Sink Gets Dual-Role Source's Extended Capabilities" shows an example sequence between a Source and a Sink when the Dual-Role Power Sink gets the Dual-Role Power Source's Extended Capabilities as a Sink.

Figure 8.75 Dual-Role Sink Gets Dual-Role Source's Extended Capabilities



[Table 8.102, "Steps for a Dual-Role Sink getting Dual-Role Source Extended Capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.75, "Dual-Role Sink Gets Dual-Role Source's Extended Capabilities"](#) above.

Table 8.102 Steps for a Dual-Role Sink getting Dual-Role Source Extended Capabilities Sequence

Step	Dual-Role Source Port	Dual-Role Sink Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Sink_Cap_Extended Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Sink_Cap_Extended Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Sink_Cap_Extended Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Sink_Cap_Extended Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Sink_Cap_Extended Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Sink_Cap_Extended Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the Extended Capabilities as a Sink which are provided. The Policy Engine tells the Protocol Layer to form a Sink_Capabilities_Extended Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Sink_Capabilities_Extended Message.	PHY Layer appends a CRC and sends the Sink_Capabilities_Extended Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Sink_Capabilities_Extended Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.102 Steps for a Dual-Role Sink getting Dual-Role Source Extended Capabilities Sequence

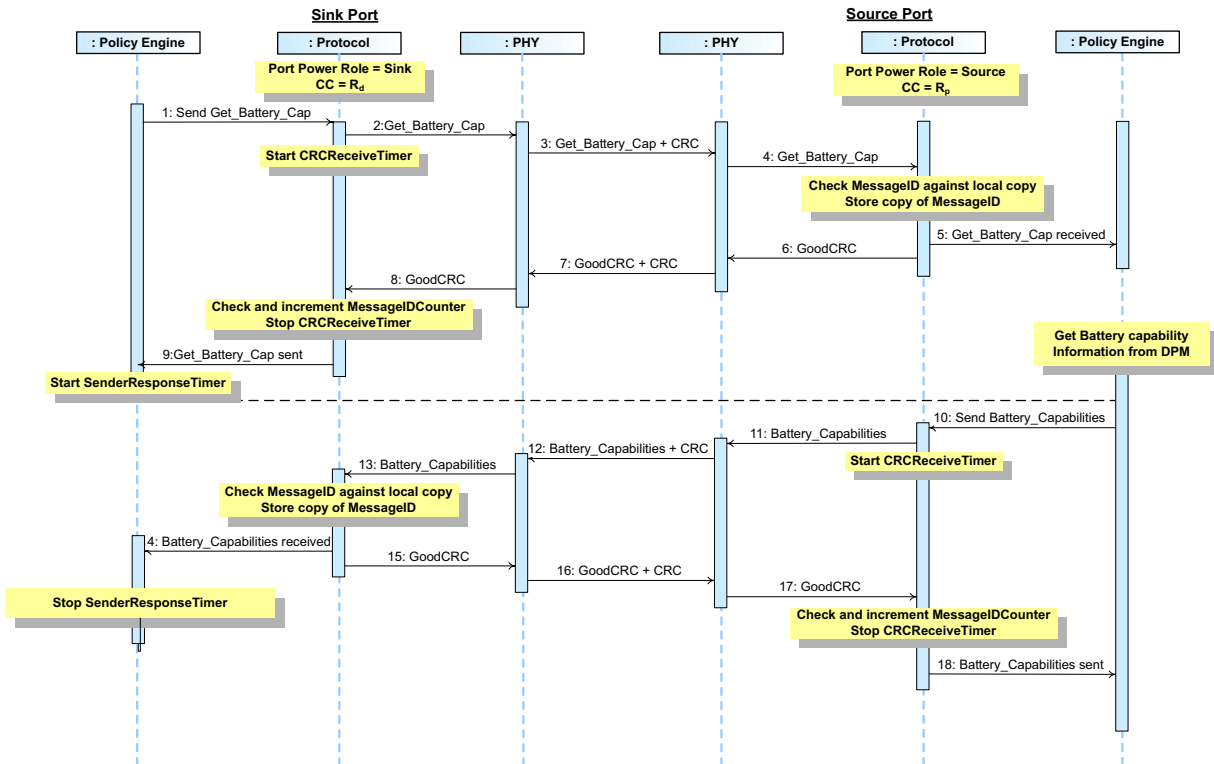
Step	Dual-Role Source Port	Dual-Role Sink Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Sink_Capabilities_Extended Message was successfully sent.</i>
<i>The Dual-Role Power Source has informed the Dual-Role Power Sink of its Extended Capabilities as a Sink.</i>		

8.3.2.11.5 Battery Capabilities and Status

8.3.2.11.5.1 Sink Gets Battery Capabilities

[Figure 8.76, "Sink Gets Source's Battery Capabilities"](#) shows an example sequence between a *Source* and a *Sink* when the *Sink* gets the *Source's* Battery capabilities for a given *Battery*.

Figure 8.76 Sink Gets Source's Battery Capabilities



[Table 8.103, "Steps for a Sink getting Source Battery capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.76, "Sink Gets Source's Battery Capabilities"](#) above.

Table 8.103 Steps for a Sink getting Source Battery capabilities Sequence

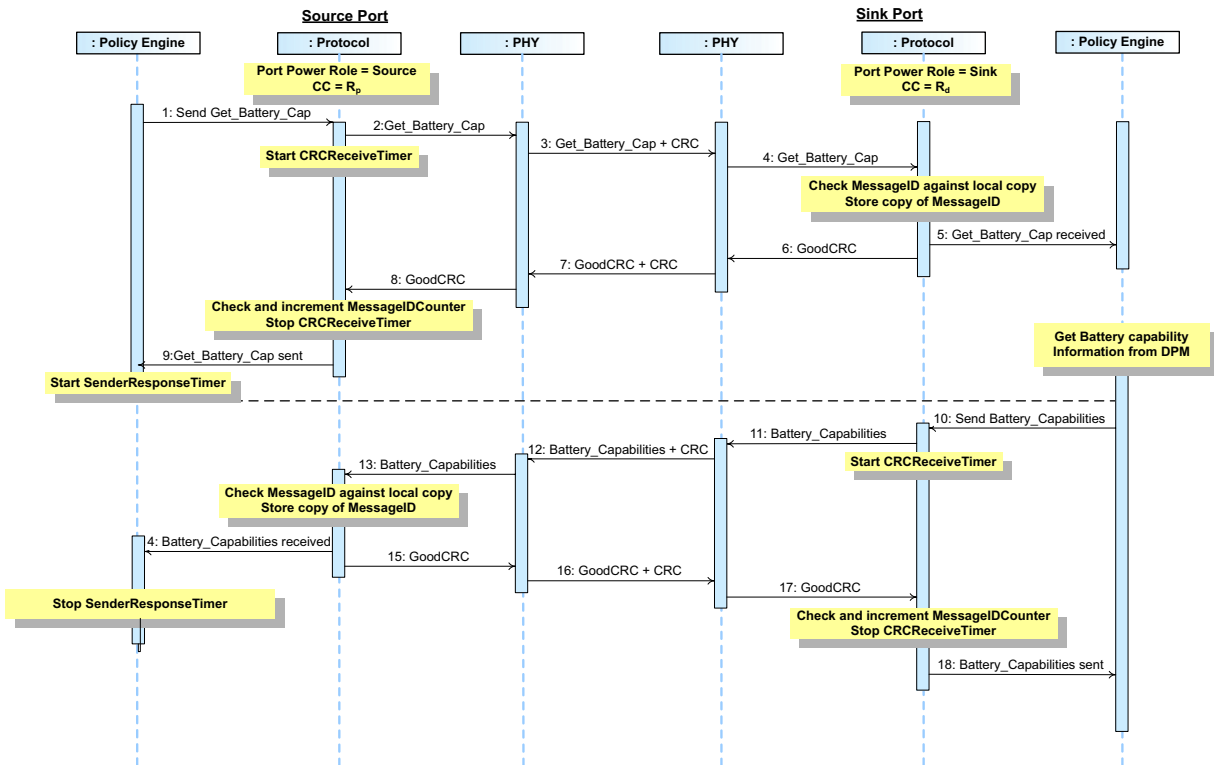
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Battery_Cap Message containing the number of the Battery for which capabilities are being requested.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Battery_Cap Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Battery_Cap Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Battery_Cap Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Battery_Cap Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Battery_Cap Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source Battery capabilities, for the requested Battery number, which are provided. The Policy Engine tells the Protocol Layer to form a Battery_Capabilities Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Battery_Capabilities Message.	PHY Layer appends a CRC and sends the Battery_Capabilities Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Battery_Capabilities Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.103 Steps for a Sink getting Source Battery capabilities Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Battery_Capabilities Message was successfully sent.</i>
The Source has informed the Sink of the Battery capabilities for the requested Battery.		

Figure 8.77, "Source Gets Sink's Battery Capabilities" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Battery capabilities for a given *Battery*.

Figure 8.77 Source Gets Sink's Battery Capabilities



[Table 8.104, "Steps for a Source getting Sink Battery capabilities Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.77, "Source Gets Sink's Battery Capabilities"](#) above.

Table 8.104 Steps for a Source getting Sink Battery capabilities Sequence

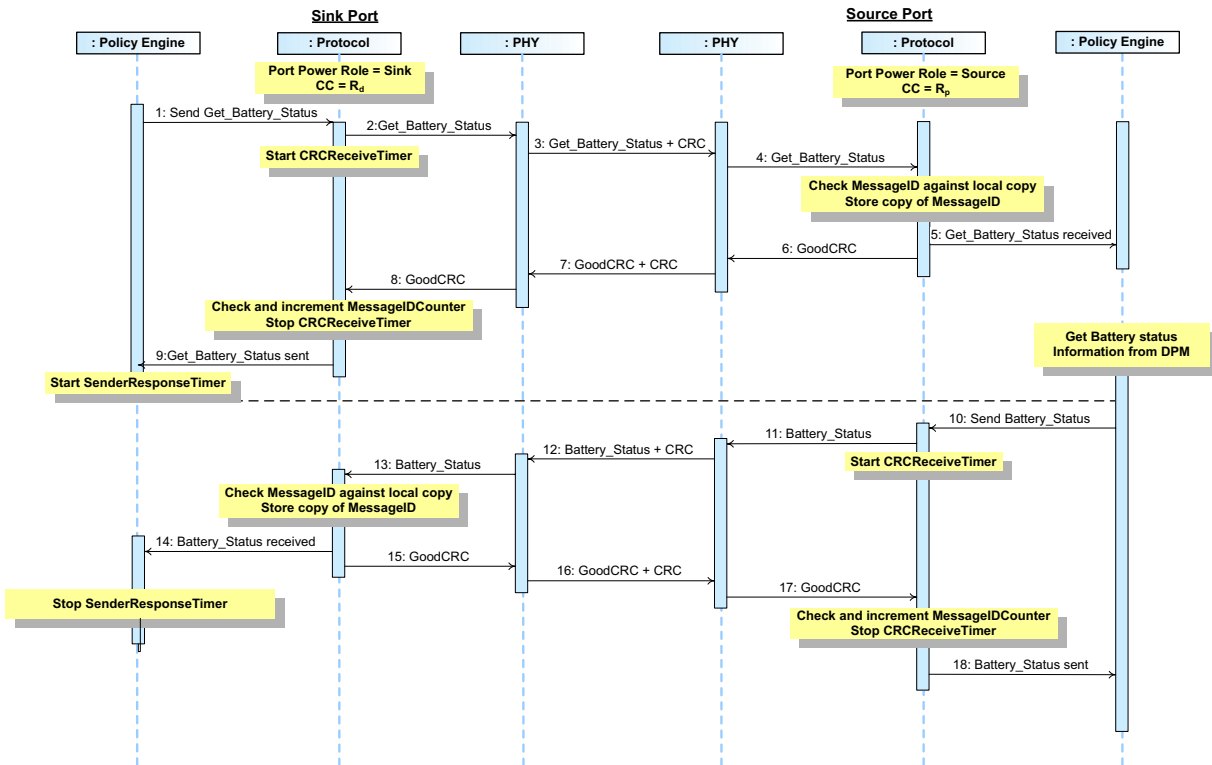
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its <i>CC</i> wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Battery_Cap Message containing the number of the <i>Battery</i> for which capabilities are being requested.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its <i>CC</i> wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Battery_Cap Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Battery_Cap Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Battery_Cap Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Battery_Cap Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Battery_Cap Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the present <i>Source Battery</i> capabilities, for the requested <i>Battery</i> number, which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Battery_Capabilities Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Battery_Capabilities Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Battery_Capabilities Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Battery_Capabilities Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.104 Steps for a Source getting Sink Battery capabilities Sequence

Step	Source Port	Sink Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Battery_Capabilities Message was successfully sent.</i>
The Sink has informed the Source of the Battery capabilities for the requested Battery.		

Figure 8.78, "Sink Gets Source's Battery Status" shows an example sequence between a *Source* and a *Sink* when the *Sink* gets the *Source*'s Battery status for a given *Battery*.

Figure 8.78 Sink Gets Source's Battery Status



[Table 8.105, "Steps for a Sink getting Source Battery status Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.78, "Sink Gets Source's Battery Status"](#) above.

Table 8.105 Steps for a Sink getting Source Battery status Sequence

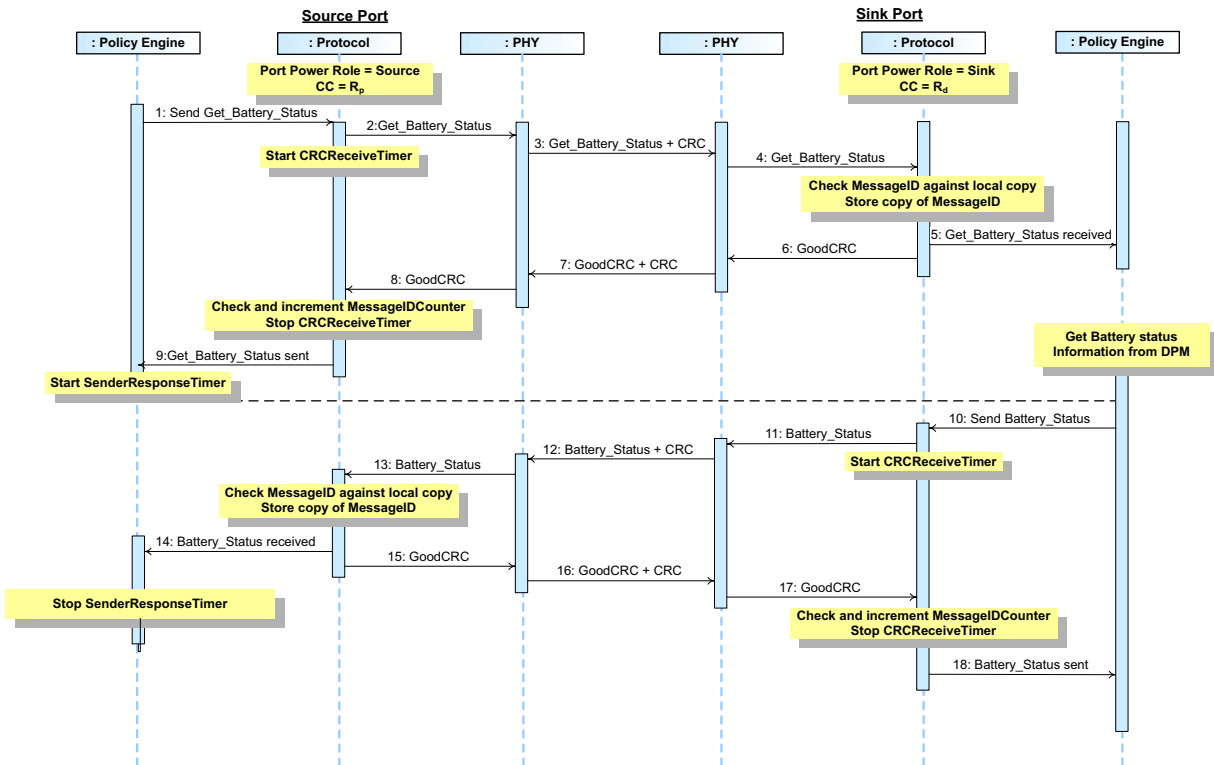
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Battery_Status Message containing the number of the Battery for which status is being requested.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Battery_Status Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Battery_Status Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Battery_Status Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Battery_Status Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Battery_Status Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source Battery status, for the requested Battery number, which are provided. The Policy Engine tells the Protocol Layer to form a Battery_Status Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Battery_Status Message.	PHY Layer appends a CRC and sends the Battery_Status Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Battery_Status Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.105 Steps for a Sink getting Source Battery status Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Battery_Status Message was successfully sent.</i>
The Source has informed the Sink of the Battery status for the requested Battery.		

Figure 8.79, "Source Gets Sink's Battery Status" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Battery status for a given *Battery*.

Figure 8.79 Source Gets Sink's Battery Status



[Table 8.106, "Steps for a Source getting Sink Battery status Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.79, "Source Gets Sink's Battery Status"](#) above.

Table 8.106 Steps for a Source getting Sink Battery status Sequence

Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its <i>CC</i> wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Battery_Status Message containing the number of the <i>Battery</i> for which status is being requested.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its <i>CC</i> wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Battery_Status Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Battery_Status Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Battery_Status Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Battery_Status Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Battery_Status Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the present <i>Source Battery</i> status, for the requested <i>Battery</i> number, which are provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Battery_Status Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Battery_Status Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Battery_Status Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Battery_Status Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.106 Steps for a Source getting Sink Battery status Sequence

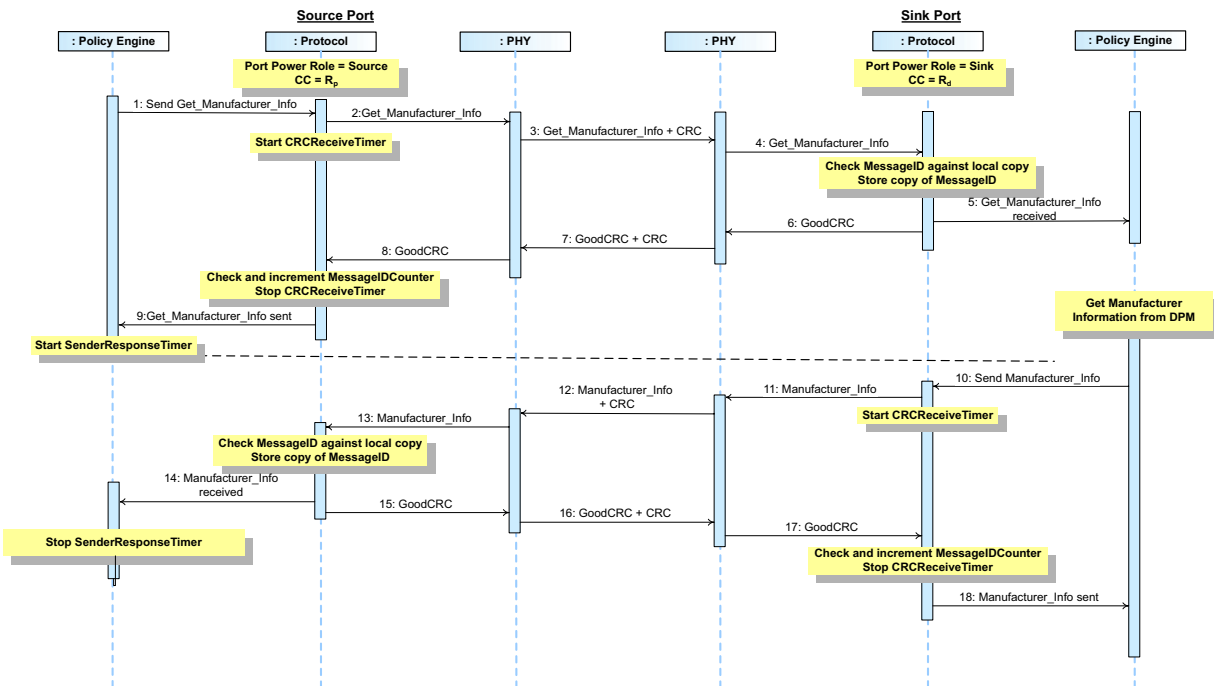
Step	Source Port	Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC <i>Message</i> .	<i>PHY Layer</i> receives GoodCRC <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Battery_Status <i>Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the <i>Battery</i> status for the requested <i>Battery</i> .		

8.3.2.11.6 Manufacturer Information

8.3.2.11.6.1 Source Gets Port Manufacturer Information from a Sink

Figure 8.80, "Source Gets Sink's Port Manufacturer Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Manufacturer information for the *Port*.

Figure 8.80 Source Gets Sink's Port Manufacturer Information



[Table 8.107, "Steps for a Source getting Sink's Port Manufacturer Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.80, "Source Gets Sink's Port Manufacturer Information"](#) above.

Table 8.107 Steps for a Source getting Sink's Port Manufacturer Information Sequence

Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Manufacturer_Info Message with a request for Port information.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Manufacturer_Info Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Manufacturer_Info Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Manufacturer_Info Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Manufacturer_Info Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Manufacturer_Info Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the Port's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Manufacturer_Info Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Manufacturer_Info Message.	<i>PHY Layer</i> appends a CRC and sends the Manufacturer_Info Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Manufacturer_Info Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	

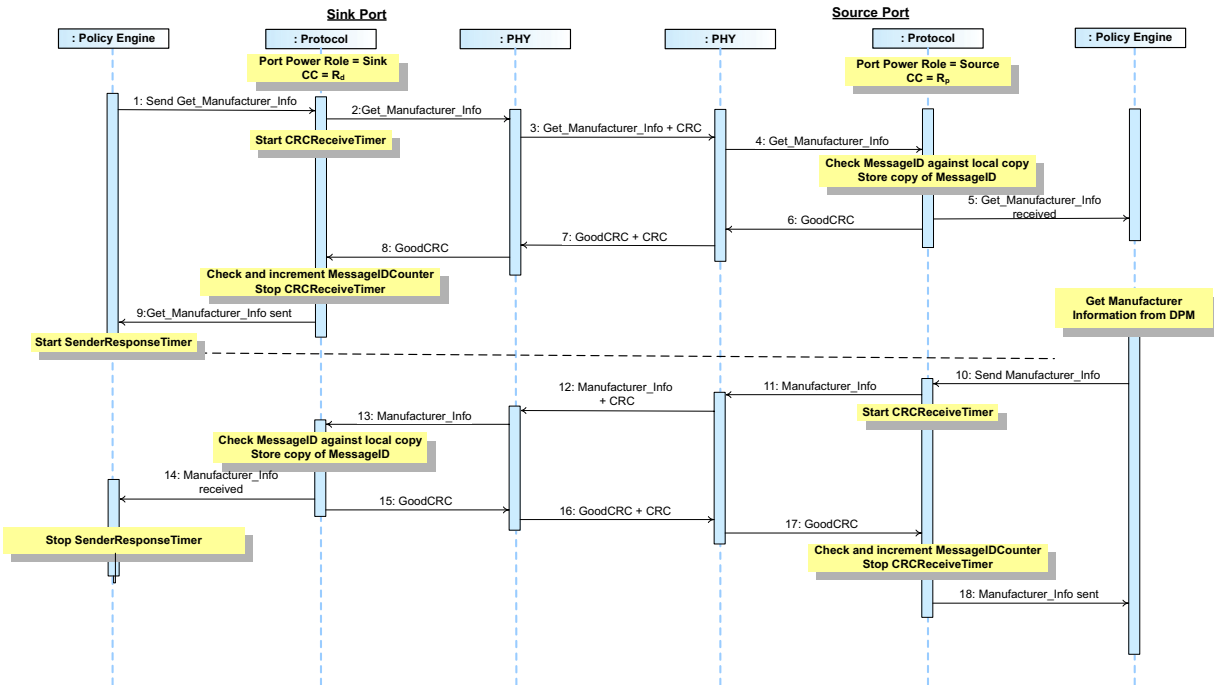
Table 8.107 Steps for a Source getting Sink's Port Manufacturer Information Sequence

Step	Source Port	Sink Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Manufacturer_Info Message was successfully sent.
The Sink has informed the Source of the manufacturer information for the Port.		

Sink Gets Port Manufacturer Information from a Source

Figure 8.81, "Sink Gets Source's Port Manufacturer Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Manufacturer information for the *Port*.

Figure 8.81 Sink Gets Source's Port Manufacturer Information



[Table 8.108, "Steps for a Source getting Sink's Port Manufacturer Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.81, "Sink Gets Source's Port Manufacturer Information"](#) above.

Table 8.108 Steps for a Source getting Sink's Port Manufacturer Information Sequence

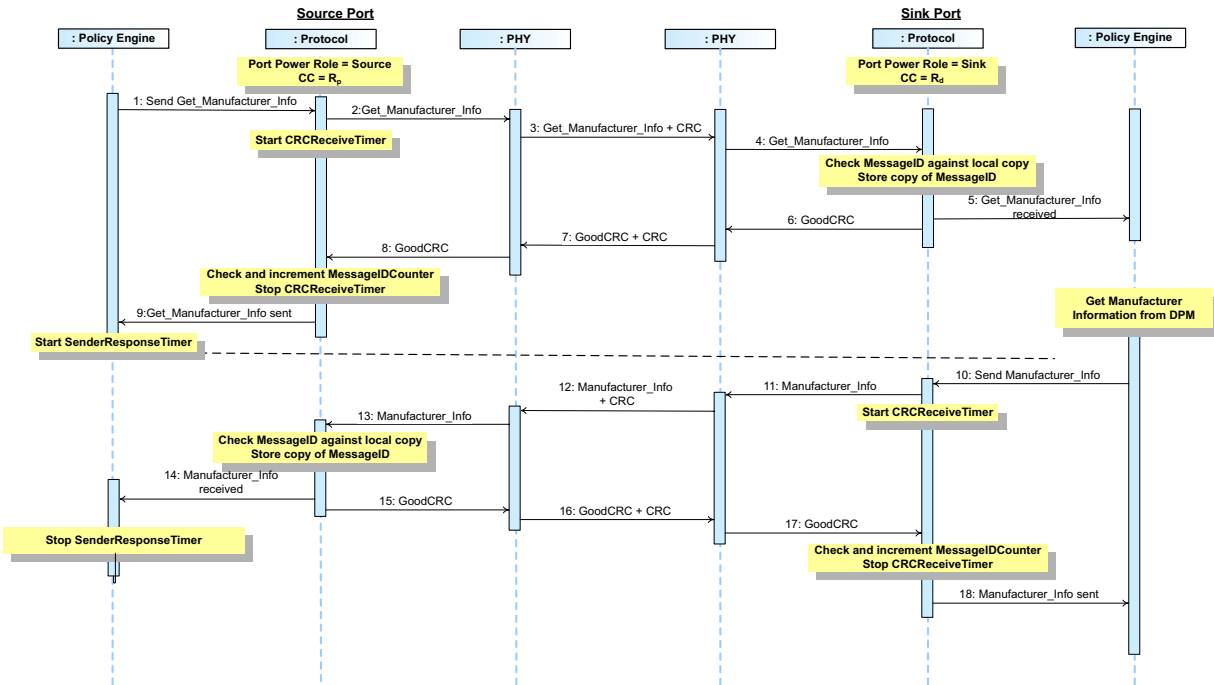
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Manufacturer_Info Message with a request for Port information.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Manufacturer_Info Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Manufacturer_Info Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Manufacturer_Info Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Manufacturer_Info Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Manufacturer_Info Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the Port's manufacturer information which is provided. The Policy Engine tells the Protocol Layer to form a Manufacturer_Info Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Manufacturer_Info Message.	PHY Layer appends a CRC and sends the Manufacturer_Info Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Manufacturer_Info Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	

Table 8.108 Steps for a Source getting Sink's Port Manufacturer Information Sequence

Step	Sink Port	Source Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Manufacturer_Info Message was successfully sent.
The Sink has informed the Source of the manufacturer information for the Port.		

Figure 8.82, "Source Gets Sink's Battery Manufacturer Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Manufacturer information for one of its Batteries.

Figure 8.82 Source Gets Sink's Battery Manufacturer Information



[Table 8.109, "Steps for a Source getting Sink's Battery Manufacturer Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.82, "Source Gets Sink's Battery Manufacturer Information"](#) above.

Table 8.109 Steps for a Source getting Sink's Battery Manufacturer Information Sequence

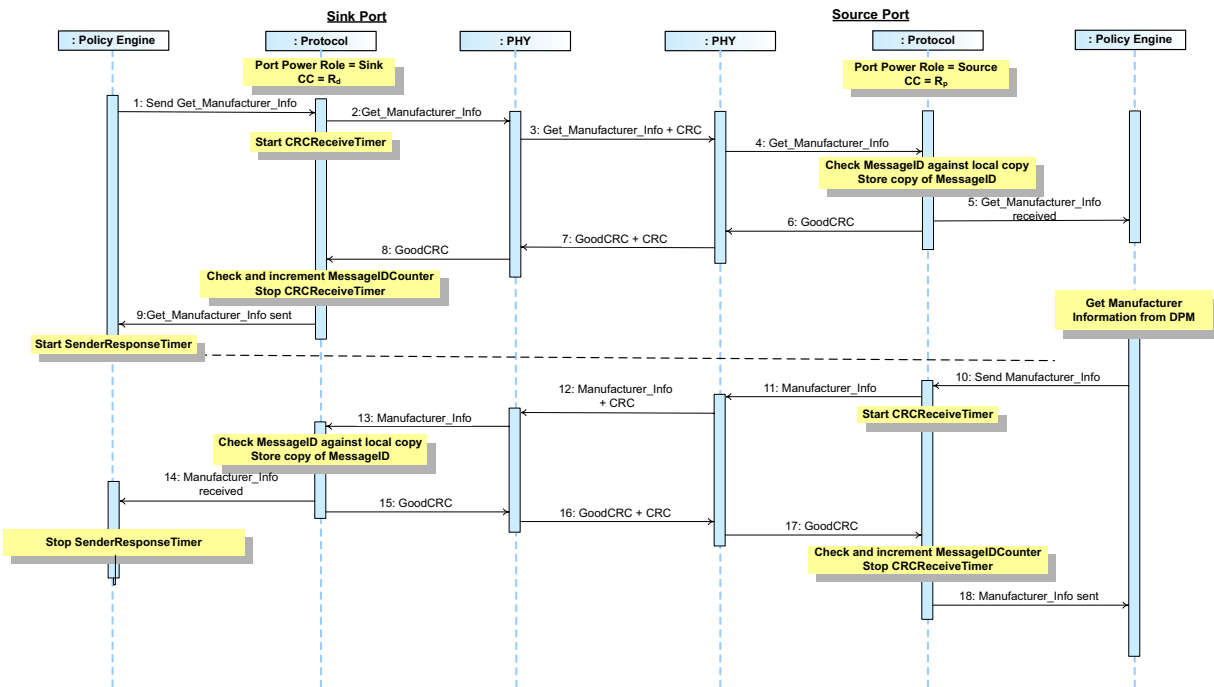
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its <i>CC</i> wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Manufacturer_Info Message with a request for <i>Battery</i> information for a given <i>Battery</i> .	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its <i>CC</i> wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Manufacturer_Info Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Manufacturer_Info Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Manufacturer_Info Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Manufacturer_Info Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Manufacturer_Info Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the <i>Battery</i> 's manufacturer information for a given <i>Battery</i> which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Manufacturer_Info Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Manufacturer_Info Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Manufacturer_Info Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Manufacturer_Info Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	

Table 8.109 Steps for a Source getting Sink's Battery Manufacturer Information Sequence

Step	Source Port	Sink Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Manufacturer_Info Message was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the manufacturer information for the requested <i>Battery</i> .		

Figure 8.83, "Sink Gets Source's Battery Manufacturer Information" shows an example sequence between a Source and a Sink when the Source gets the Sink's Manufacturer information for the Port.

Figure 8.83 Sink Gets Source's Battery Manufacturer Information



[Table 8.110, "Steps for a Source getting Sink's Battery Manufacturer Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.83, "Sink Gets Source's Battery Manufacturer Information"](#) above.

Table 8.110 Steps for a Source getting Sink's Battery Manufacturer Information Sequence

Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Manufacturer_Info Message with a request for Battery information for a given Battery.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Manufacturer_Info Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Manufacturer_Info Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Manufacturer_Info Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Manufacturer_Info Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Manufacturer_Info Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the Battery's manufacturer information for a given Battery which is provided. The Policy Engine tells the Protocol Layer to form a Manufacturer_Info Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Manufacturer_Info Message.	PHY Layer appends a CRC and sends the Manufacturer_Info Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Manufacturer_Info Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	

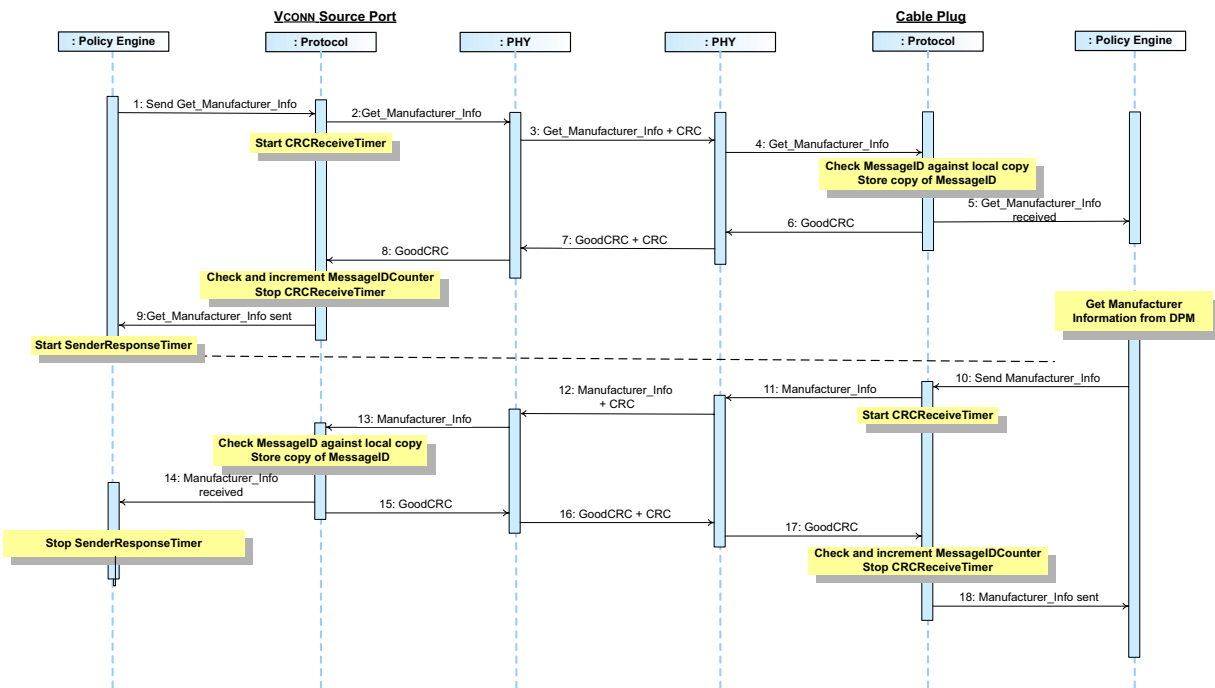
Table 8.110 Steps for a Source getting Sink's Battery Manufacturer Information Sequence

Step	Sink Port	Source Port
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Manufacturer_Info Message was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the manufacturer information for the requested <i>Battery</i> .		

VCONN Source Gets Manufacturer Information from a Cable Plug

[Figure 8.84, "VCONN.Source Gets Cable Plug's Manufacturer Information"](#) shows an example sequence between a VCONN Source (Source or Sink) and a Cable Plug when the VCONN Source gets the Cable Plug's Manufacturer information.

Figure 8.84 VCONN Source Gets Cable Plug's Manufacturer Information



[Table 8.111, "Steps for a VCONN Source getting Sink's Port Manufacturer Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.84, "VCONN Source Gets Cable Plug's Manufacturer Information"](#) above.

Table 8.111 Steps for a VCONN Source getting Sink's Port Manufacturer Information Sequence

Step	VCONN Source	Cable Plug
1	The <i>Port</i> is currently acting as the <i>VCONN Source</i> . <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Manufacturer_Info Message with a request for <i>Port</i> information.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Manufacturer_Info Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Manufacturer_Info Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Manufacturer_Info Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Manufacturer_Info Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Manufacturer_Info Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the <i>Cable Plug's</i> manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Manufacturer_Info Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Manufacturer_Info Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Manufacturer_Info Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Manufacturer_Info Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.111 Steps for a VCONN Source getting Sink's Port Manufacturer Information Sequence

Step	VCONN Source	Cable Plug
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Manufacturer_Info Message</i> was successfully sent.
The <i>Cable Plug</i> has informed the <i>Source</i> of its manufacturer information.		

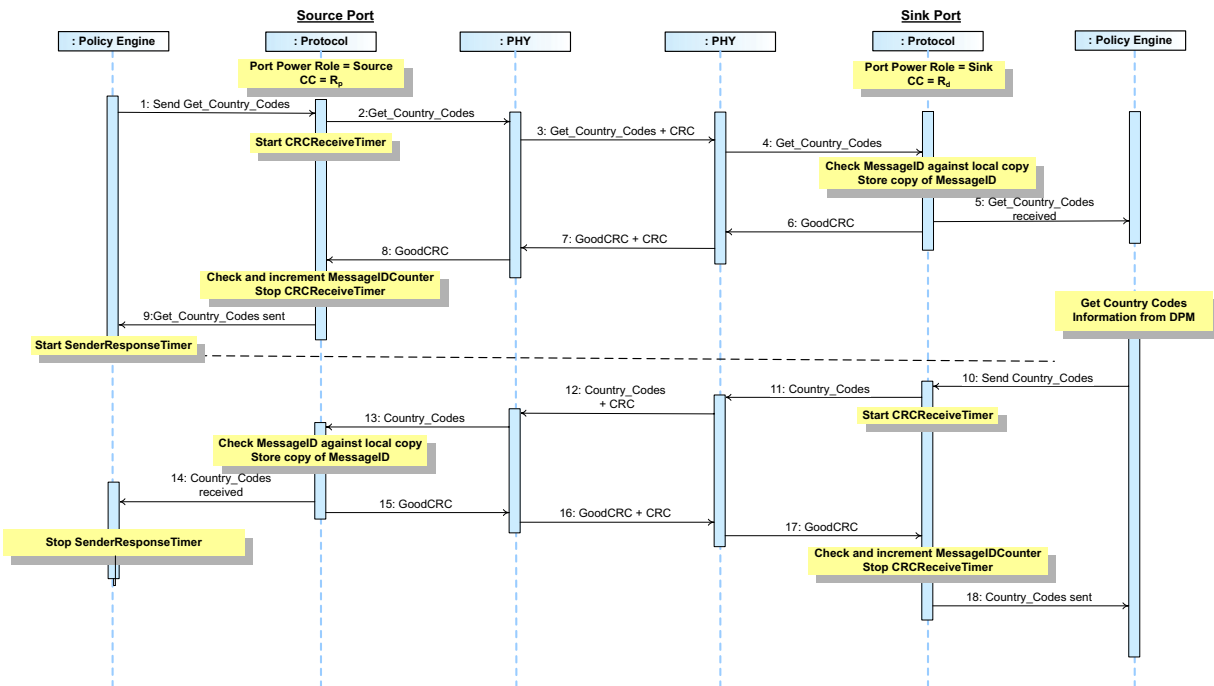
8.3.2.11.7 Country Codes

8.3.2.11.7.1

8.3.2.12.7.1 Source Gets Country Codes from a Sink

Figure 8.85, "Source Gets Sink's Country Codes" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Country Codes.

Figure 8.85 Source Gets Sink's Country Codes



[Table 8.112, "Steps for a Source getting Country Codes Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.85, "Source Gets Sink's Country Codes"](#) above.

Table 8.112 Steps for a Source getting Country Codes Sequence

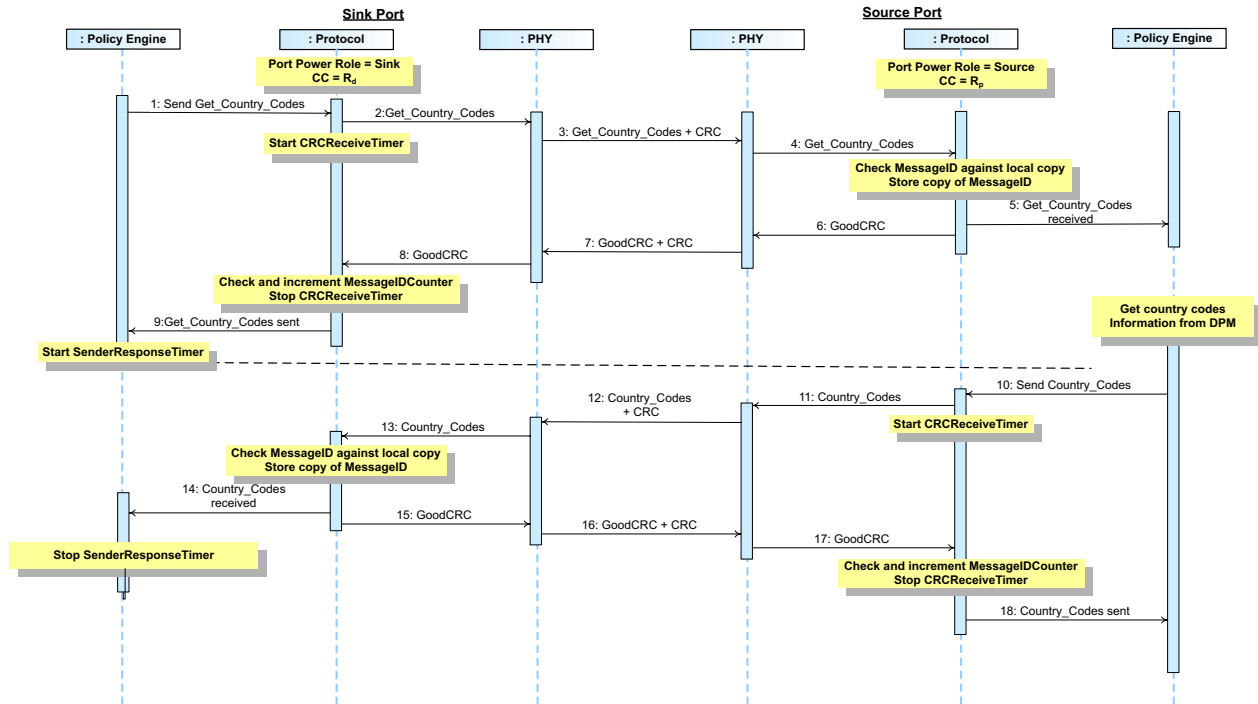
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its <i>CC</i> wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Country_Codes Message with a request for Port information.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its <i>CC</i> wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Country_Codes Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Country_Codes Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Country_Codes Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Country_Codes Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Country_Codes Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the Port's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Country_Codes Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Country_Codes Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Country_Codes Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Country_Codes Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.112 Steps for a Source getting Country Codes Sequence

Step	Source Port	Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Country_Codes Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the country codes.		

Figure 8.86, "Sink Gets Source's Country Codes" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s country codes.

Figure 8.86 Sink Gets Source's Country Codes



[Table 8.113, "Steps for a Source getting Sink's Country Codes Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.86, "Sink Gets Source's Country Codes"](#) above.

Table 8.113 Steps for a Source getting Sink's Country Codes Sequence

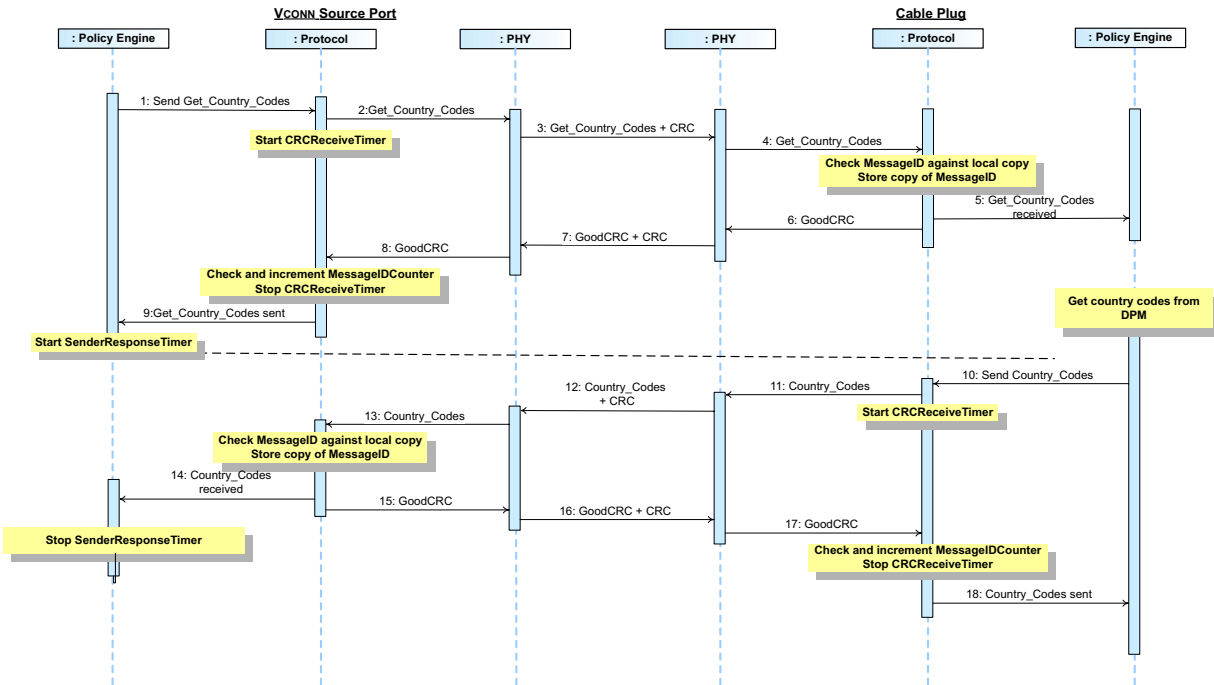
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Country_Codes Message with a request for Port information.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Country_Codes Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Country_Codes Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Country_Codes Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Country_Codes Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Country_Codes Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the Port's manufacturer information which is provided. The Policy Engine tells the Protocol Layer to form a Country_Codes Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Country_Codes Message.	PHY Layer appends a CRC and sends the Country_Codes Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Country_Codes Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.113 Steps for a Source getting Sink's Country Codes Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Country_Codes Message was successfully sent.</i>
The Sink has informed the Source of the country codes.		

Figure 8.87, "VCONN Source Gets Cable Plug's Country Codes" shows an example sequence between a VCONN Source (Source or Sink) and a Cable Plug when the VCONN Source gets the Cable Plug's Country Codes.

Figure 8.87 VCONN Source Gets Cable Plug's Country Codes



[Table 8.114, "Steps for a VCONN Source getting Sink's Country Codes Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.87, "VCONN Source Gets Cable Plug's Country Codes"](#) above.

Table 8.114 Steps for a VCONN Source getting Sink's Country Codes Sequence

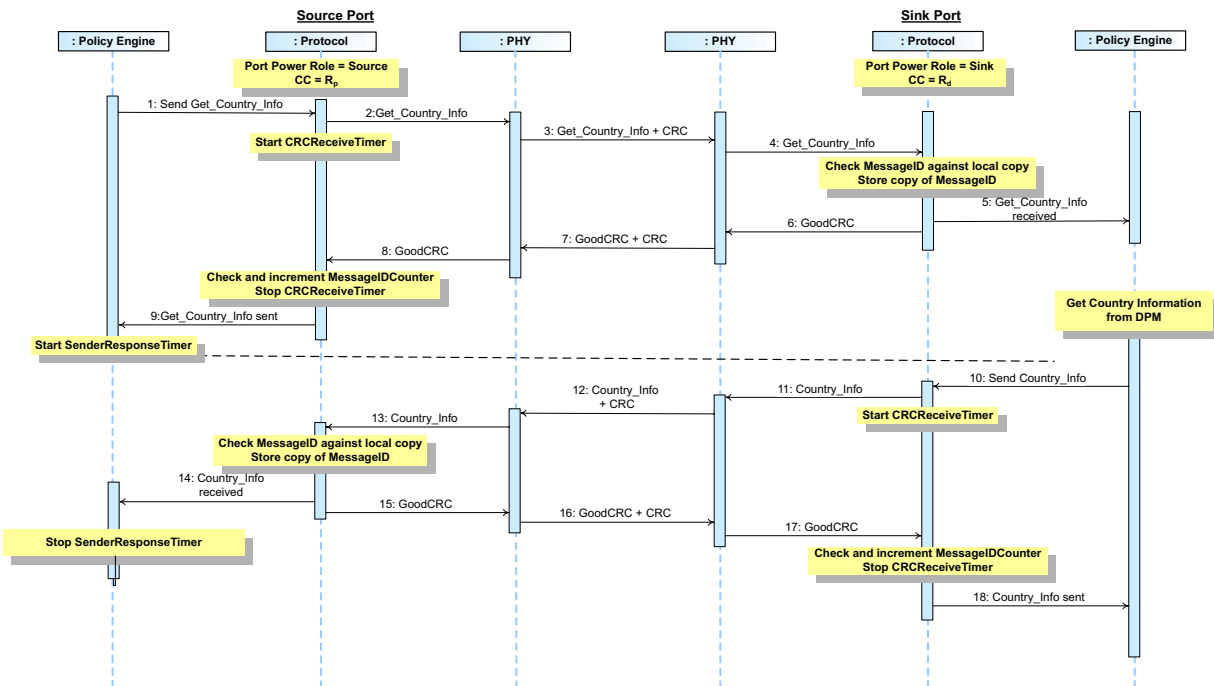
Step	VCONN Source	Cable Plug
1	The <i>Port</i> is currently acting as the <i>VCONN Source</i> . <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>Get_Country_Codes</i> Message with a request for <i>Port</i> information.	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Get_Country_Codes</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Get_Country_Codes</i> Message and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Get_Country_Codes</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Get_Country_Codes</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Get_Country_Codes</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the <i>Cable Plug's</i> manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a <i>Country_Codes</i> Message.
11		<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Country_Codes</i> Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Country_Codes</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Country_Codes</i> Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the <i>SenderResponseTimer</i> .	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	

Table 8.114 Steps for a VCONN Source getting Sink's Country Codes Sequence

Step	VCONN Source	Cable Plug
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Country_Codes Message was successfully sent.</i>
The Cable Plug has informed the Source of its country codes.		

Figure 8.88, "Source Gets Sink's Country Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s country information.

Figure 8.88 Source Gets Sink's Country Information



[Table 8.115, "Steps for a Source getting Country Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.88, "Source Gets Sink's Country Information"](#) above.

Table 8.115 Steps for a Source getting Country Information Sequence

Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its <i>CC</i> wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Country_Info Message with a request for Port information for a specific Country Code.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its <i>CC</i> wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Country_Info Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Country_Info Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Country_Info Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Country_Info Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Country_Info Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the Port's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Country_Info Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Country_Info Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Country_Info Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Country_Info Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

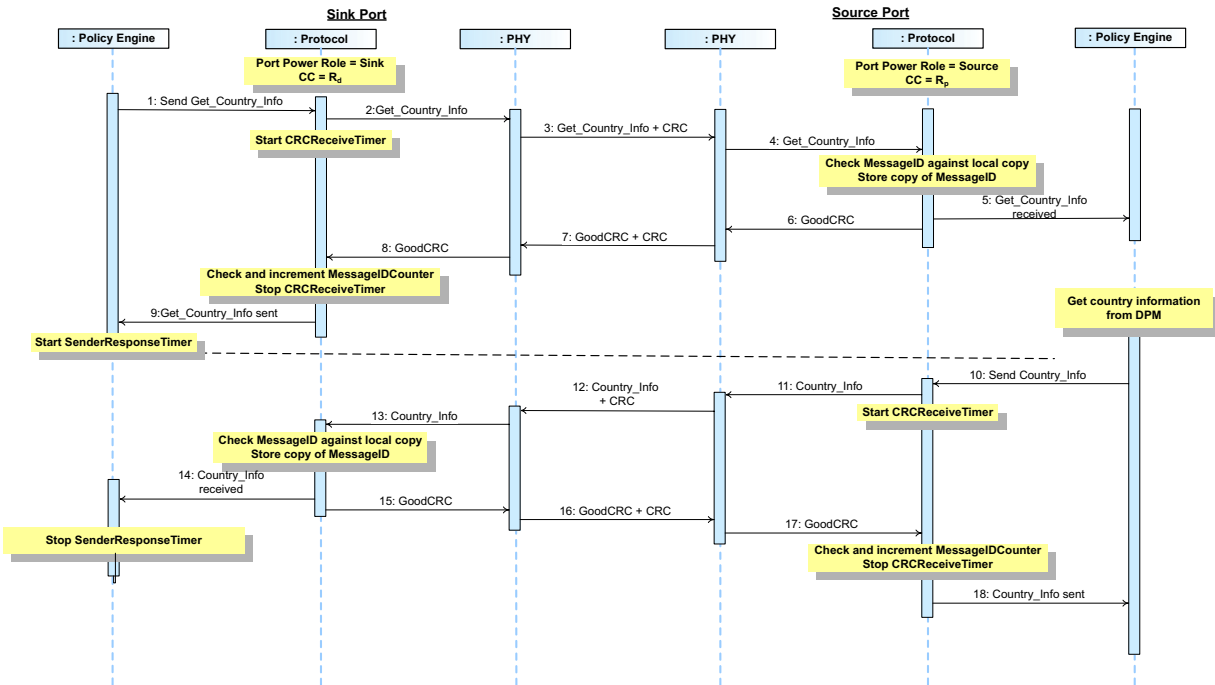
Table 8.115 Steps for a Source getting Country Information Sequence

Step	Source Port	Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Country_Info Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the country information.		

Sink Gets Country Information from a Source

Figure 8.89. "Sink Gets Source's Country Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s country codes.

Figure 8.89 Sink Gets Source's Country Information



[Table 8.116, "Steps for a Source getting Sink's Country Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.89, "Sink Gets Source's Country Information"](#) above.

Table 8.116 Steps for a Source getting Sink's Country Information Sequence

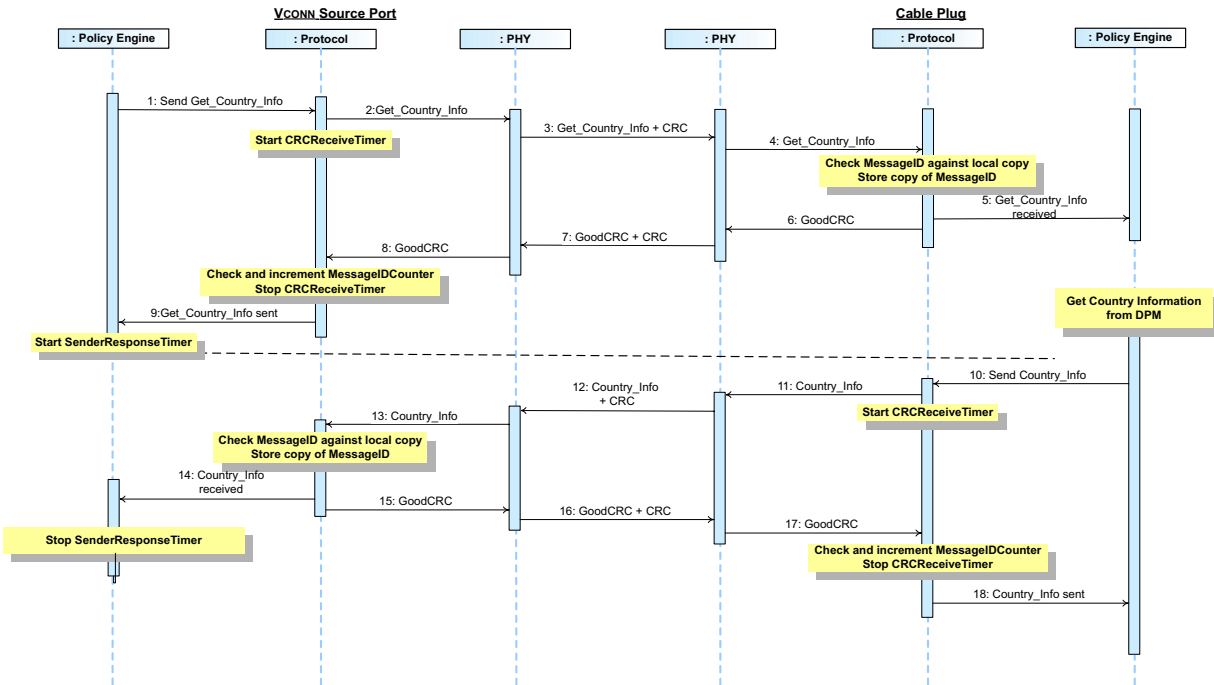
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Country_Info Message with a request for Port information for a specific country code.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Country_Info Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Country_Info Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Country_Info Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Country_Info Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Country_Info Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the Port's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Country_Info Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Country_Info Message.	<i>PHY Layer</i> appends a CRC and sends the Country_Info Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Country_Info Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.116 Steps for a Source getting Sink's Country Information Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Country_Info Message was successfully sent.</i>
The Sink has informed the Source of the country information.		

Figure 8.90, "VCONN Source Gets Cable Plug's Country Information" shows an example sequence between a VCONN Source (Source or Sink) and a Cable Plug when the VCONN Source gets the Cable Plug's country information.

Figure 8.90 VCONN Source Gets Cable Plug's Country Information



[Table 8.117, "Steps for a VCONN Source getting Sink's Country Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.90, "VCONN Source Gets Cable Plug's Country Information"](#) above.

Table 8.117 Steps for a VCONN Source getting Sink's Country Information Sequence

Step	VCONN Source	Cable Plug
1	The <i>Port</i> is currently acting as the <i>VCONN Source</i> . <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a <i>Get_Country_Info</i> Message with a request for <i>Port</i> information for a specific country code.	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Get_Country_Info</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Get_Country_Info</i> Message and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Get_Country_Info</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Get_Country_Info</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Get_Country_Info</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the <i>Cable Plug's</i> manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a <i>Country_Info</i> Message.
11		<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Country_Info</i> Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Country_Info</i> Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Country_Info</i> Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the <i>SenderResponseTimer</i> .	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	

Table 8.117 Steps for a VCONN Source getting Sink's Country Information Sequence

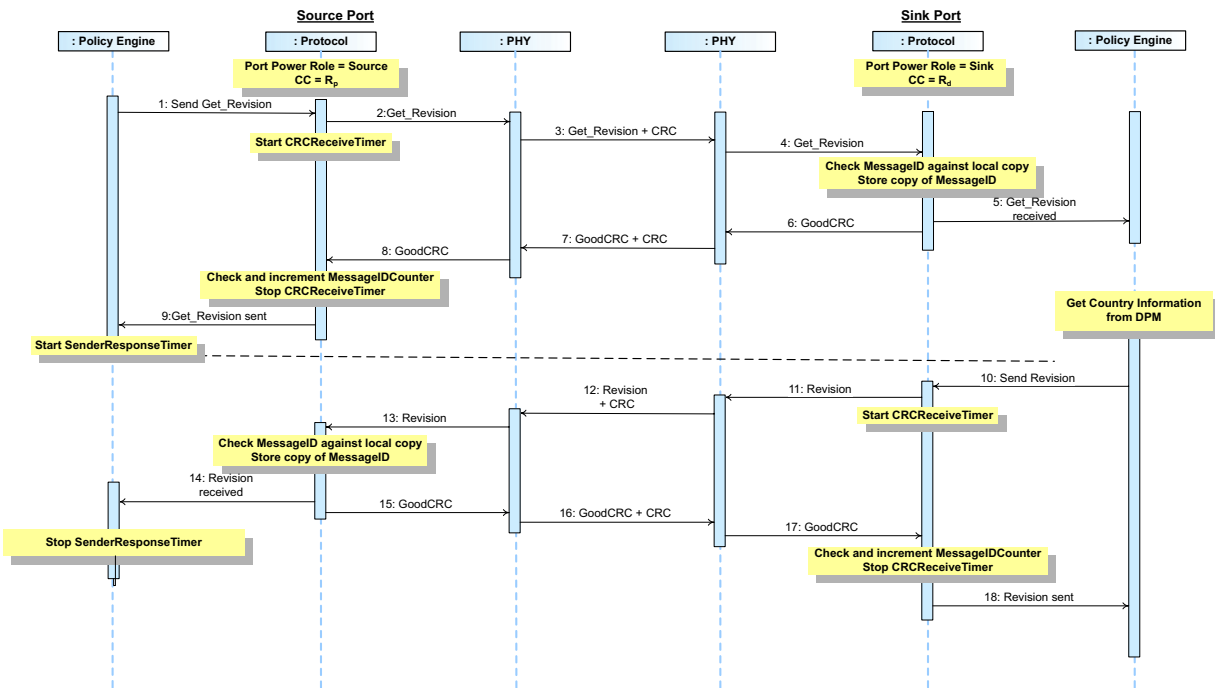
Step	VCONN Source	Cable Plug
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Country_Info Message was successfully sent.</i>
The Cable Plug has informed the Source of its country information.		

8.3.2.11.9 Revision Information

8.3.2.11.9.1 Source Gets Revision Information from a Sink

Figure 8.91, "Source Gets Sink's Revision Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Revision information.

Figure 8.91 Source Gets Sink's Revision Information



[Table 8.118, "Steps for a Source getting Revision Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.91, "Source Gets Sink's Revision Information"](#) above.

Table 8.118 Steps for a Source getting Revision Information Sequence

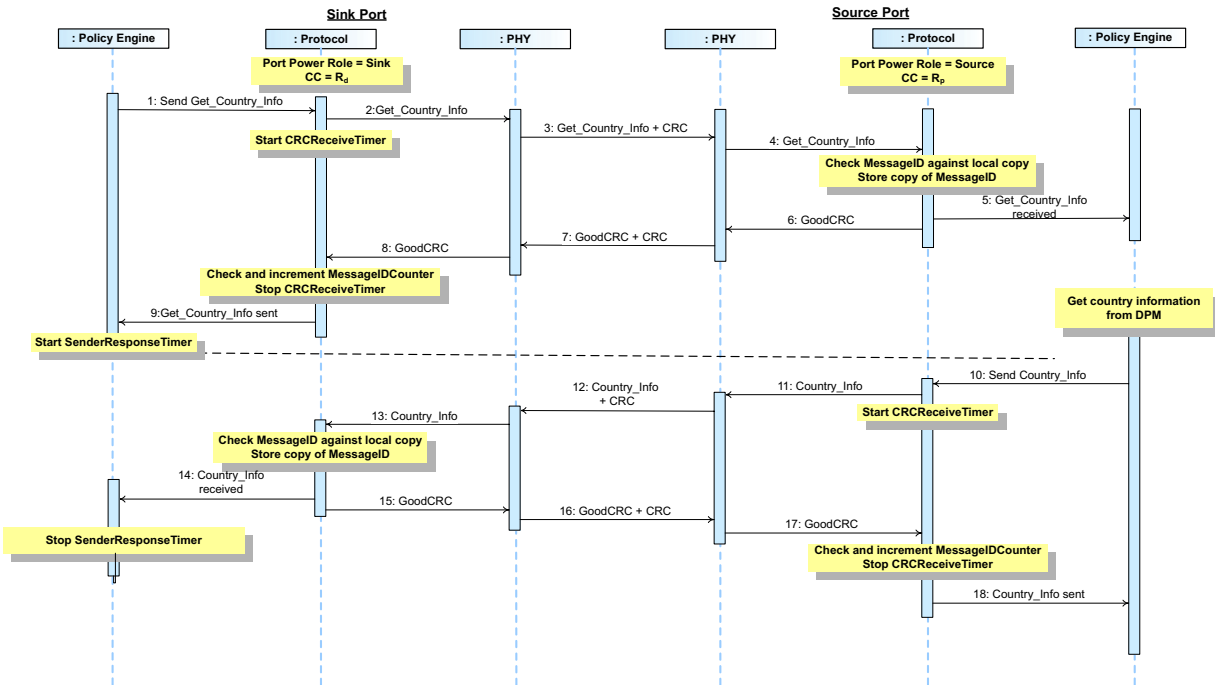
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its <i>CC</i> wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Revision Message with a request for Port information for a specific Revision Code.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its <i>CC</i> wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Revision Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Revision Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Revision Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Revision Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Revision Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the Port's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Revision Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Revision_Info Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Revision Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Revision Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.118 Steps for a Source getting Revision Information Sequence

Step	Source Port	Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Revision Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the Revision information.		

Figure 8.92, "Sink Gets Source's Revision Information" shows an example sequence between a *Source* and a *Sink* when the *Source* gets the *Sink*'s Revision codes.

Figure 8.92 Sink Gets Source's Revision Information



[Table 8.119, "Steps for a Source getting Sink's Revision Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.92, "Sink Gets Source's Revision Information"](#) above.

Table 8.119 Steps for a Source getting Sink's Revision Information Sequence

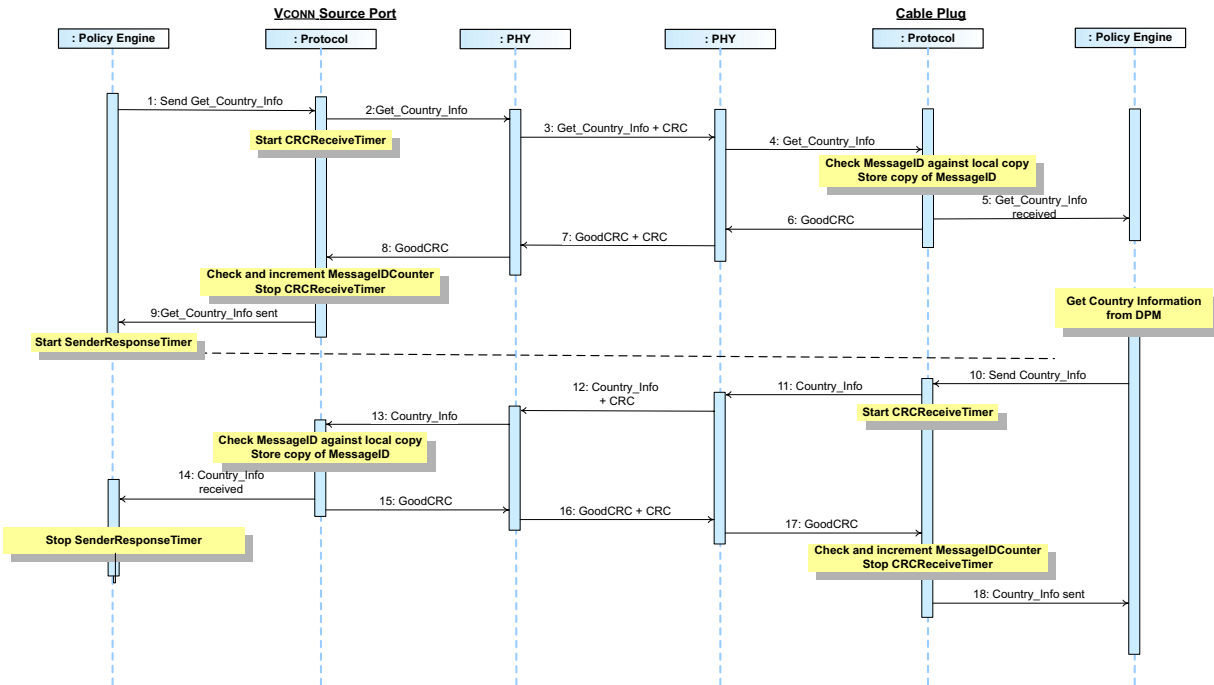
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Revision Message with a request for Port information for a specific Revision code.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Revision Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Revision Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Revision Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Revision Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Revision Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the DPM for the Port's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Revision Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Revision_Info Message.	<i>PHY Layer</i> appends a CRC and sends the Revision Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Revision Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.119 Steps for a Source getting Sink's Revision Information Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Revision Message</i> was successfully sent.
The <i>Sink</i> has informed the <i>Source</i> of the Revision information.		

Figure 8.93, "*VCONN Source Gets Cable Plug's Revision Information*" shows an example sequence between a VCONN Source (Source or Sink) and a Cable Plug when the VCONN Source gets the Cable Plug's Revision information.

Figure 8.93 VCONN Source Gets Cable Plug's Revision Information



[Table 8.120, "Steps for a VCONN Source getting Sink's Revision Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.93, "VCONN Source Gets Cable Plug's Revision Information"](#) above.

Table 8.120 Steps for a VCONN Source getting Sink's Revision Information Sequence

Step	VCONN Source	Cable Plug
1	The <i>Port</i> is currently acting as the <i>VCONN Source</i> . <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Revision Message with a request for <i>Port</i> information for a specific Revision code.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Get_Revision Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Revision Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Get_Revision Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Revision Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Revision Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the <i>Cable Plug</i> 's manufacturer information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Revision Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the <i>CRC</i> it calculated with the one sent to verify the Revision Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Revision Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Revision Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.120 Steps for a VCONN Source getting Sink's Revision Information Sequence

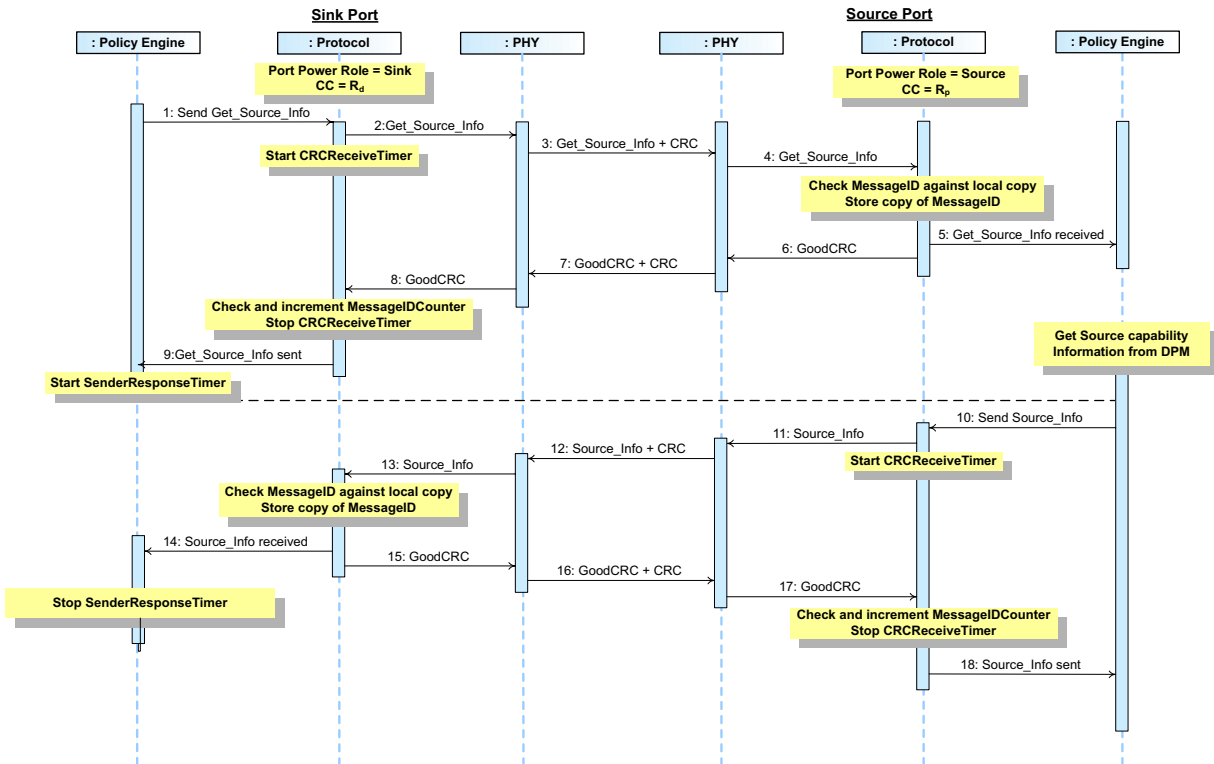
Step	VCONN Source	Cable Plug
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Revision Message</i> was successfully sent.
The <i>Cable Plug</i> has informed the <i>Source</i> of its Revision information.		

8.3.2.11.10 Source Information

8.3.2.11.10.1 Sink Gets Source Information

[Figure 8.94, "Sink Gets Source's Information"](#) shows an example sequence between a *Source* and a *Sink* when the *Sink* gets the *Source*'s information.

Figure 8.94 Sink Gets Source's Information



[Table 8.121, "Steps for a Sink getting Source Information Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.94, "Sink Gets Source's Information"](#) above.

Table 8.121 Steps for a Sink getting Source Information Sequence

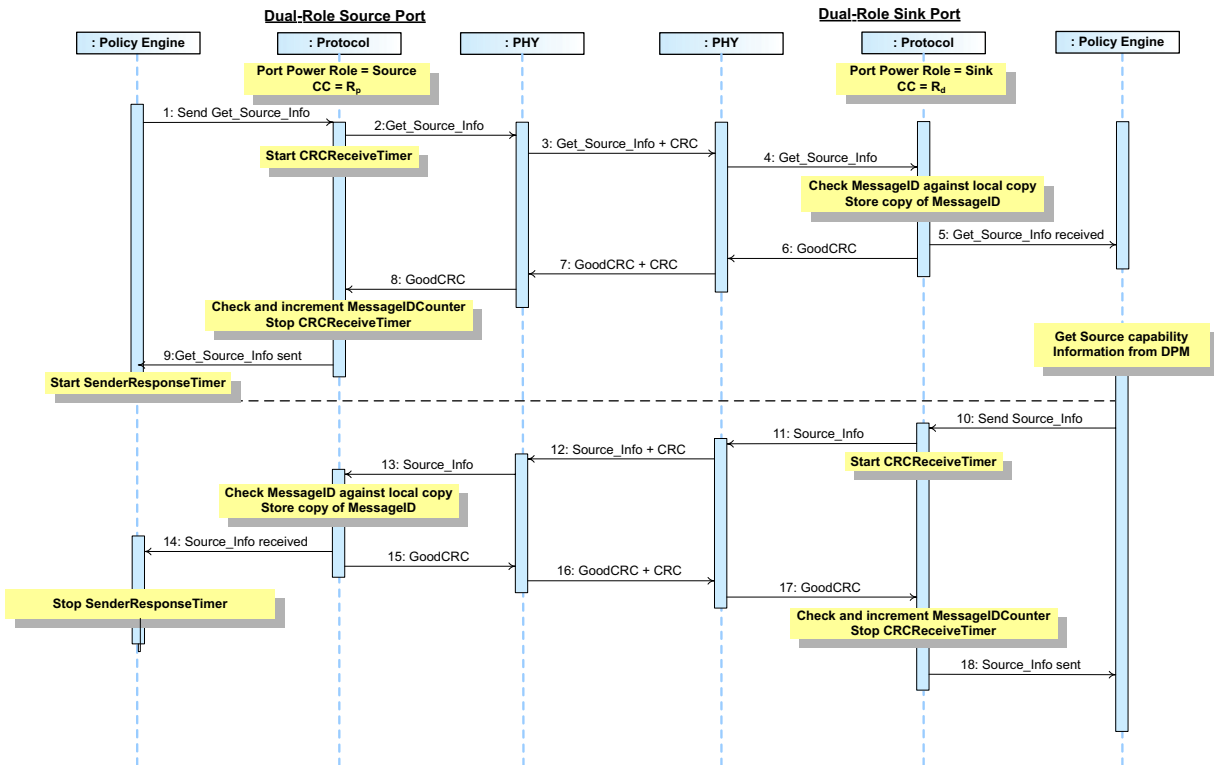
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Get_Source_Info Message.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Get_Source_Info Message. Starts CRCReceiveTimer .	PHY Layer receives the Get_Source_Info Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Get_Source_Info Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Get_Source_Info Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Get_Source_Info Message was successfully sent. Policy Engine starts SenderResponseTimer .	
10		Policy Engine requests the DPM for the present Source information which is provided. The Policy Engine tells the Protocol Layer to form a Source_Info Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Source_Info Message.	PHY Layer appends a CRC and sends the Source_Info Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Source_Info Message information to the Policy Engine that consumes it.	
14	The Policy Engine stops the SenderResponseTimer .	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.121 Steps for a Sink getting Source Information Sequence

Step	Sink Port	Source Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Info Message</i> was successfully sent.
The <i>Source</i> has provided the <i>Sink</i> with its information.		

Figure 8.95, "Dual-Role Source Gets Dual-Role Sink's Information as a Source" shows an example sequence between a Dual-Role Power Source and a Dual-Role Power Sink when the Source gets the Sink's Information as a Source.

Figure 8.95 Dual-Role Source Gets Dual-Role Sink's Information as a Source



[Table 8.122, "Steps for a Dual-Role Source getting Dual-Role Sink's Information as a Source Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.95, "Dual-Role Source Gets Dual-Role Sink's Information as a Source"](#) above.

Table 8.122 Steps for a Dual-Role Source getting Dual-Role Sink's Information as a Source Sequence

Step	Dual-Role Source Port	Dual-Role Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Get_Source_Info Message.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Get_Source_Info Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Get_Source_Info Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Get_Source_Info Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Get_Source_Info Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Get_Source_Info Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the present <i>Source</i> information which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Source_Info Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Source_Info Message.	<i>PHY Layer</i> appends a CRC and sends the Source_Info Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Source_Info Message information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the SenderResponseTimer .	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.122 Steps for a Dual-Role Source getting Dual-Role Sink's Information as a Source Sequence

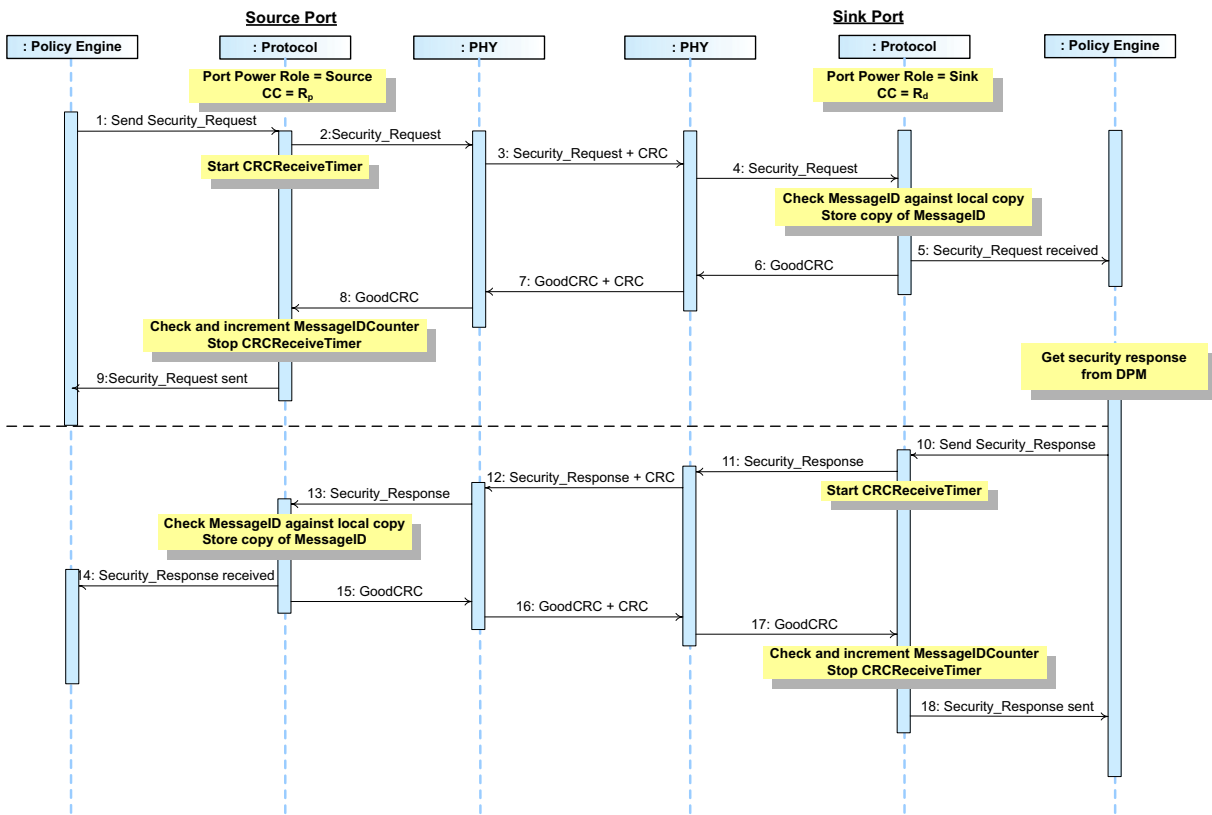
Step	Dual-Role Source Port	Dual-Role Sink Port
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Source_Info Message</i> was successfully sent.
The <i>Dual-Role Power Sink</i> has provided the <i>Dual-Role Power Source</i> with its information.		

8.3.2.12 Security

8.3.2.12.1 Source requests security exchange with Sink

[Figure 8.96, "Source requests security exchange with Sink"](#) shows an example sequence for a security exchange between a *Source* and a *Sink*.

Figure 8.96 Source requests security exchange with Sink



[Table 8.123, "Steps for a Source requesting a security exchange with a Sink Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.96, "Source requests security exchange with Sink"](#) above.

Table 8.123 Steps for a Source requesting a security exchange with a Sink Sequence

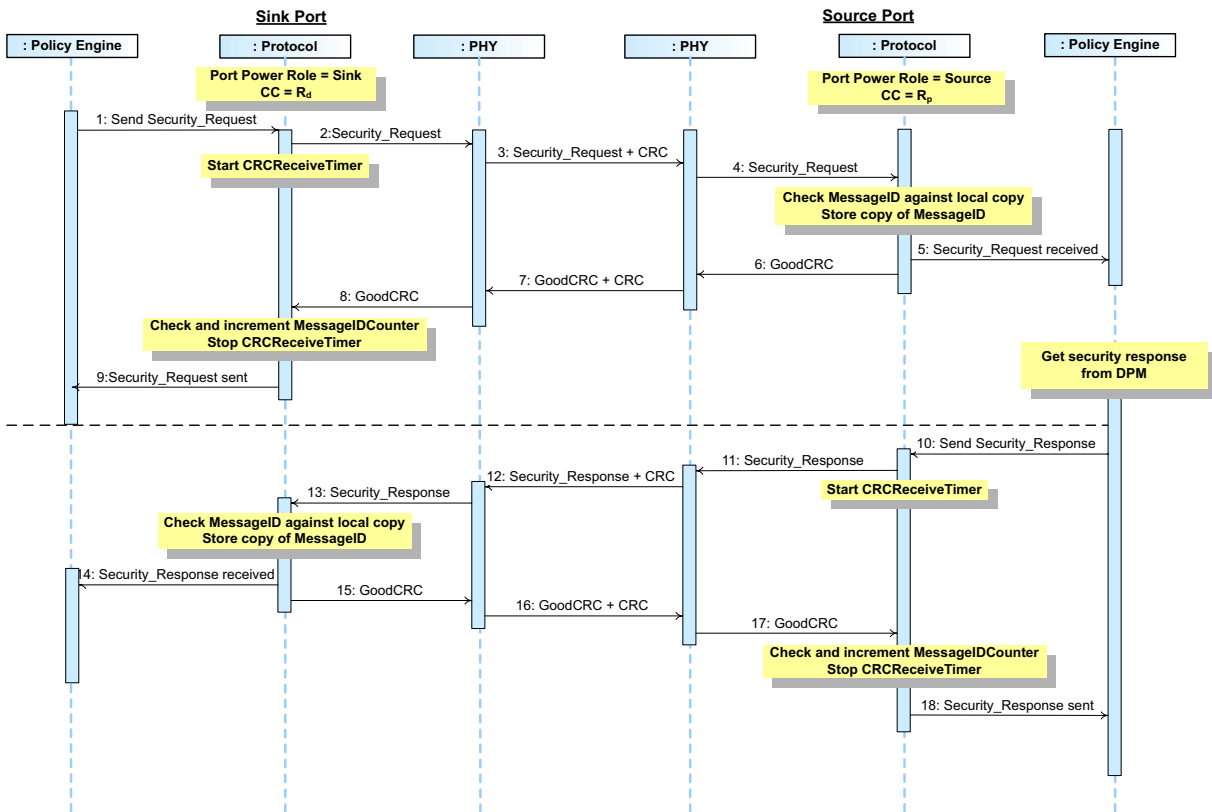
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. Policy Engine directs the Protocol Layer to send a Security_Request Message using a Payload supplied by the DPM.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Security_Request Message. Starts CRCReceiveTimer .	PHY Layer receives the Security_Request Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Security_Request Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Security_Request Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Security_Request Message was successfully sent.	
10		Policy Engine requests the DPM for the response to the security request which is provided. The Policy Engine tells the Protocol Layer to form a Security_Response Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Security_Response Message.	PHY Layer appends a CRC and sends the Security_Response Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Security_Response Message information to the Policy Engine that consumes it.	
14	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
15	PHY Layer appends a CRC and sends the GoodCRC Message.	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.123 Steps for a Source requesting a security exchange with a Sink Sequence

Step	Source Port	Sink Port
16		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Security_Response Message</i> was successfully sent.
The security exchange is complete.		

Figure 8.97, "Sink requests security exchange with Source" shows an example sequence for a security exchange between a Sink and a Source.

Figure 8.97 Sink requests security exchange with Source



[Table 8.124, "Steps for a Sink requesting a security exchange with a Source Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.97, "Sink requests security exchange with Source"](#) above.

Table 8.124 Steps for a Sink requesting a security exchange with a Source Sequence

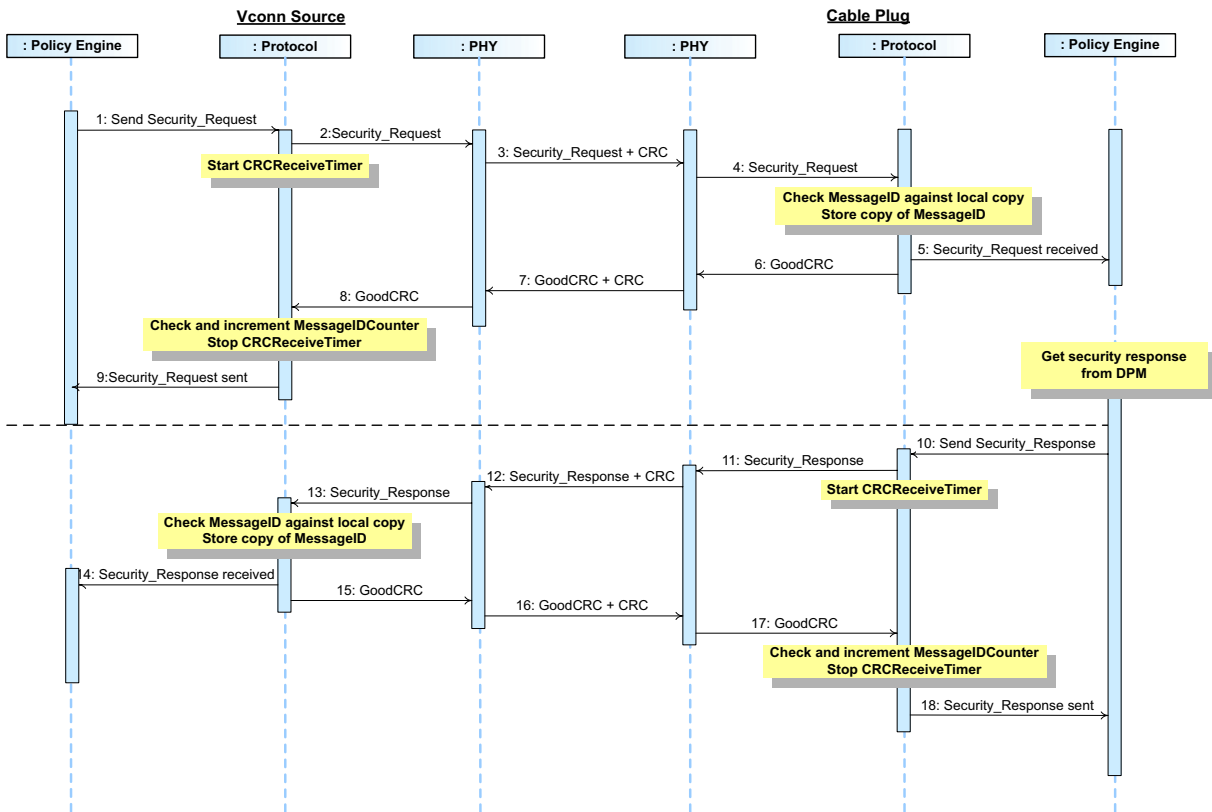
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Security_Request Message using a Payload supplied by the DPM.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Security_Request Message. Starts CRCReceiveTimer .	PHY Layer receives the Security_Request Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Security_Request Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Security_Request Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Security_Request Message was successfully sent.	
10		Policy Engine requests the DPM for the response to the security request which is provided. The Policy Engine tells the Protocol Layer to form a Security_Response Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Security_Response Message.	PHY Layer appends a CRC and sends the Security_Response Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Security_Response Message information to the Policy Engine that consumes it.	
14	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.124 Steps for a Sink requesting a security exchange with a Source Sequence

Step	Sink Port	Source Port
15	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
16		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Security_Response Message</i> was successfully sent.
The security exchange is complete.		

Figure 8.98, "VCONN Source requests security exchange with Cable Plug" shows an example sequence for a security exchange between a VCONN Source and a Cable Plug.

Figure 8.98 VCONN Source requests security exchange with Cable Plug



[Table 8.125, "Steps for a VCONN Source requesting a security exchange with a Cable Plug Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.98, "VCONN Source requests security exchange with Cable Plug"](#) above.

Table 8.125 Steps for a VCONN Source requesting a security exchange with a Cable Plug Sequence

Step	VCONN Source	Cable Plug
1	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Security_Request Message using a <i>Payload</i> supplied by the <i>DPM</i> .	
2	<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Security_Request Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Security_Request Message and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Security_Request Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Security_Request Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Security_Request Message was successfully sent.	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the response to the security request which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Security_Response Message.
11		<i>Protocol Layer</i> creates the <i>Message</i> and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the Security_Response Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Security_Response Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Security_Response Message information to the <i>Policy Engine</i> that consumes it.	
14	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
15	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

Table 8.125 Steps for a VCONN Source requesting a security exchange with a Cable Plug Sequence

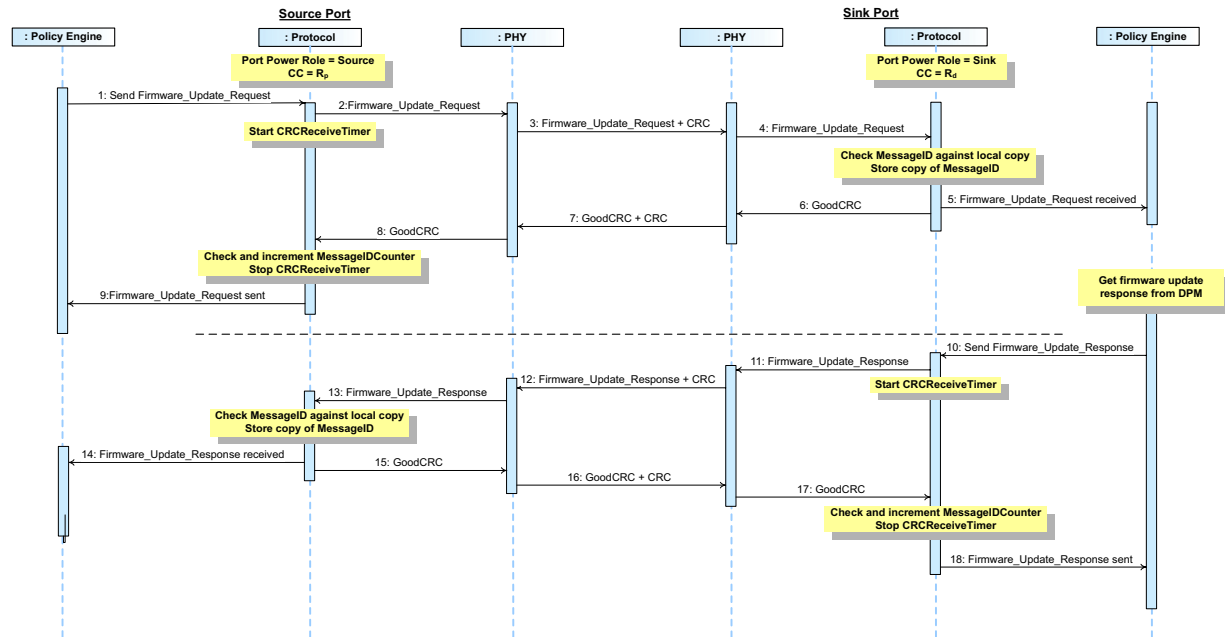
Step	VCONN Source	Cable Plug
16		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Security_Response Message</i> was successfully sent.
The security exchange is complete.		

8.3.2.13 Firmware Update

8.3.2.13.1 Source requests firmware update exchange with Sink

Figure 8.99, "Source requests firmware update exchange with Sink" shows an example sequence for a firmware update exchange between a *Source* and a *Sink*.

Figure 8.99 Source requests firmware update exchange with Sink



[Table 8.126, "Steps for a Source requesting a firmware update exchange with a Sink Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.99, "Source requests firmware update exchange with Sink"](#) above.

Table 8.126 Steps for a Source requesting a firmware update exchange with a Sink Sequence

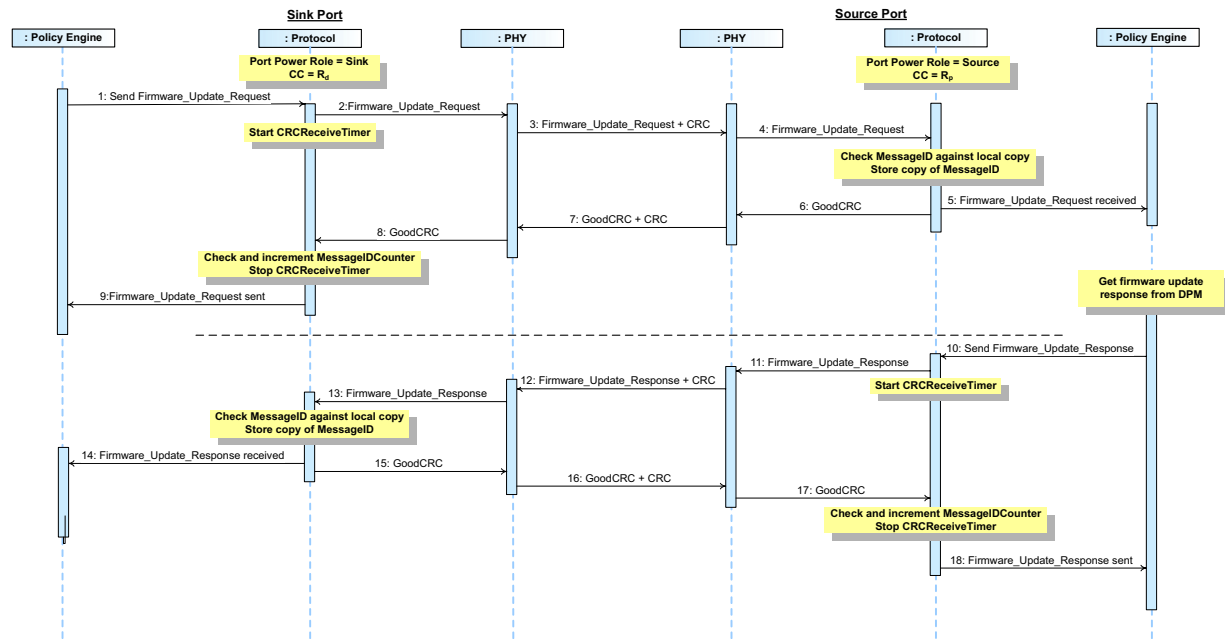
Step	Source Port	Sink Port
1	The Port has Port Power Role set to <i>Source</i> and the R_p pull up on its CC wire. Policy Engine directs the Protocol Layer to send a Firmware_Update_Request Message using a Payload supplied by the DPM.	The Port has Port Power Role set to <i>Sink</i> with the R_d pull down on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Firmware_Update_Request Message. Starts CRCReceiveTimer .	PHY Layer receives the Firmware_Update_Request Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Firmware_Update_Request Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Firmware_Update_Request Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Firmware_Update_Request Message was successfully sent.	
10		Policy Engine requests the DPM for the response to the firmware update request which is provided. The Policy Engine tells the Protocol Layer to form a Firmware_Update_Response Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Firmware_Update_Response Message.	PHY Layer appends a CRC and sends the Firmware_Update_Response Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Firmware_Update_Response Message information to the Policy Engine that consumes it.	

Table 8.126 Steps for a Source requesting a firmware update exchange with a Sink Sequence

Step	Source Port	Sink Port
14	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
15	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
16		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Firmware_Update_Response Message was successfully sent.
The firmware update exchange is complete.		

Figure 8.100, "Sink requests firmware update exchange with Source" shows an example sequence for a firmware update exchange between a Sink and a Source.

Figure 8.100 Sink requests firmware update exchange with Source



[Table 8.127, "Steps for a Sink requesting a firmware update exchange with a Source Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.100, "Sink requests firmware update exchange with Source"](#) above.

Table 8.127 Steps for a Sink requesting a firmware update exchange with a Source Sequence

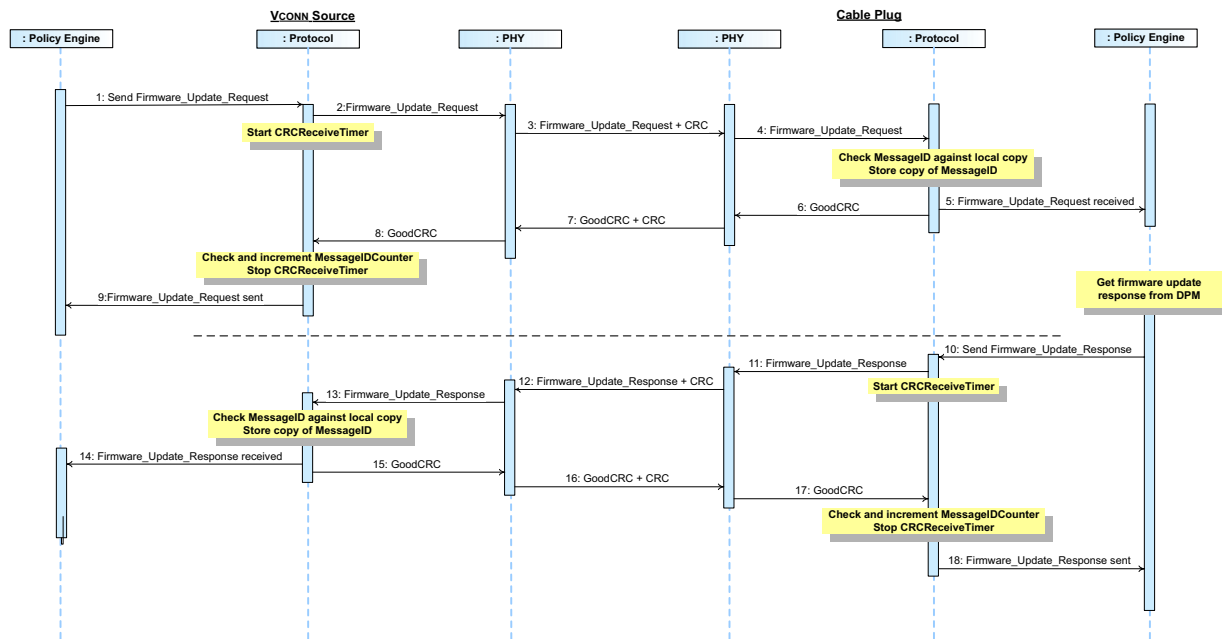
Step	Sink Port	Source Port
1	The Port has Port Power Role set to Sink with the R_d pull down on its CC wire. Policy Engine directs the Protocol Layer to send a Firmware_Update_Request Message using a Payload supplied by the DPM.	The Port has Port Power Role set to Source and the R_p pull up on its CC wire.
2	Protocol Layer creates the Message and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Firmware_Update_Request Message. Starts CRCReceiveTimer .	PHY Layer receives the Firmware_Update_Request Message and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Firmware_Update_Request Message to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Firmware_Update_Request Message information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Firmware_Update_Request Message was successfully sent.	
10		Policy Engine requests the DPM for the response to the firmware update request which is provided. The Policy Engine tells the Protocol Layer to form a Firmware_Update_Response Message.
11		Protocol Layer creates the Message and passes to PHY Layer.
12	PHY Layer receives the Message and compares the CRC it calculated with the one sent to verify the Firmware_Update_Response Message.	PHY Layer appends a CRC and sends the Firmware_Update_Response Message. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Firmware_Update_Response Message information to the Policy Engine that consumes it.	

Table 8.127 Steps for a Sink requesting a firmware update exchange with a Source Sequence

Step	Sink Port	Source Port
14	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
15	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
16		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Firmware_Update_Response Message was successfully sent.
The firmware update exchange is complete.		

Figure 8.101, "VCONN Source requests firmware update exchange with Cable Plug" shows an example sequence for a firmware update exchange between a VCONN Source and a Cable Plug.

Figure 8.101 VCONN Source requests firmware update exchange with Cable Plug



[Table 8.128, "Steps for a VCONN Source requesting a firmware update exchange with a Cable Plug Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.101, "VCONN Source requests firmware update exchange with Cable Plug"](#) above.

Table 8.128 Steps for a VCONN Source requesting a firmware update exchange with a Cable Plug Sequence

Step	VCONN Source	Cable Plug
1	<i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Firmware_Update_Request Message using a Payload supplied by the <i>DPM</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Firmware_Update_Request Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Firmware_Update_Request Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Firmware_Update_Request Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Firmware_Update_Request Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Firmware_Update_Request Message was successfully sent.	
10		<i>Policy Engine</i> requests the <i>DPM</i> for the response to the firmware update request which is provided. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Firmware_Update_Response Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Firmware_Update_Response Message.	<i>PHY Layer</i> appends a CRC and sends the Firmware_Update_Response Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Firmware_Update_Response Message information to the <i>Policy Engine</i> that consumes it.	
14	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.128 Steps for a VCONN Source requesting a firmware update exchange with a Cable Plug Sequence

Step	VCONN Source	Cable Plug
15	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
16		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Firmware_Update_Response Message</i> was successfully sent.
The firmware update exchange is complete.		

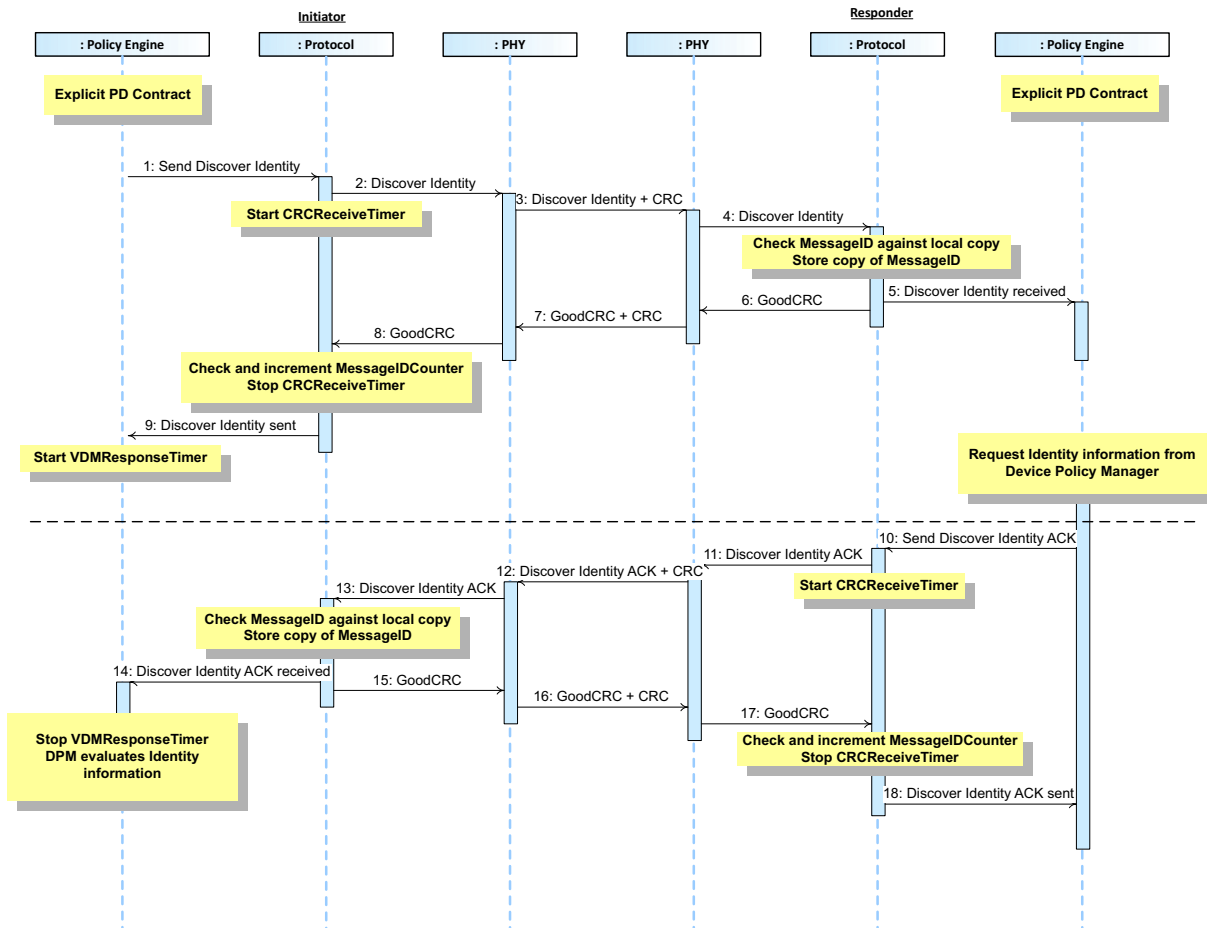
8.3.2.14 Structured VDM

8.3.2.14.1 Discover Identity

8.3.2.14.1.1 Initiator to Responder Discover Identity (ACK)

[Figure 8.102, "Initiator to Responder Discover Identity \(ACK\)"](#) shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* discovers identity information from the *Responder*.

Figure 8.102 Initiator to Responder Discover Identity (ACK)



[Table 8.129, "Steps for Initiator to UFP Discover Identity \(ACK\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.102, "Initiator to Responder Discover Identity \(ACK\)"](#) above.

Table 8.129 Steps for Initiator to UFP Discover Identity (ACK)

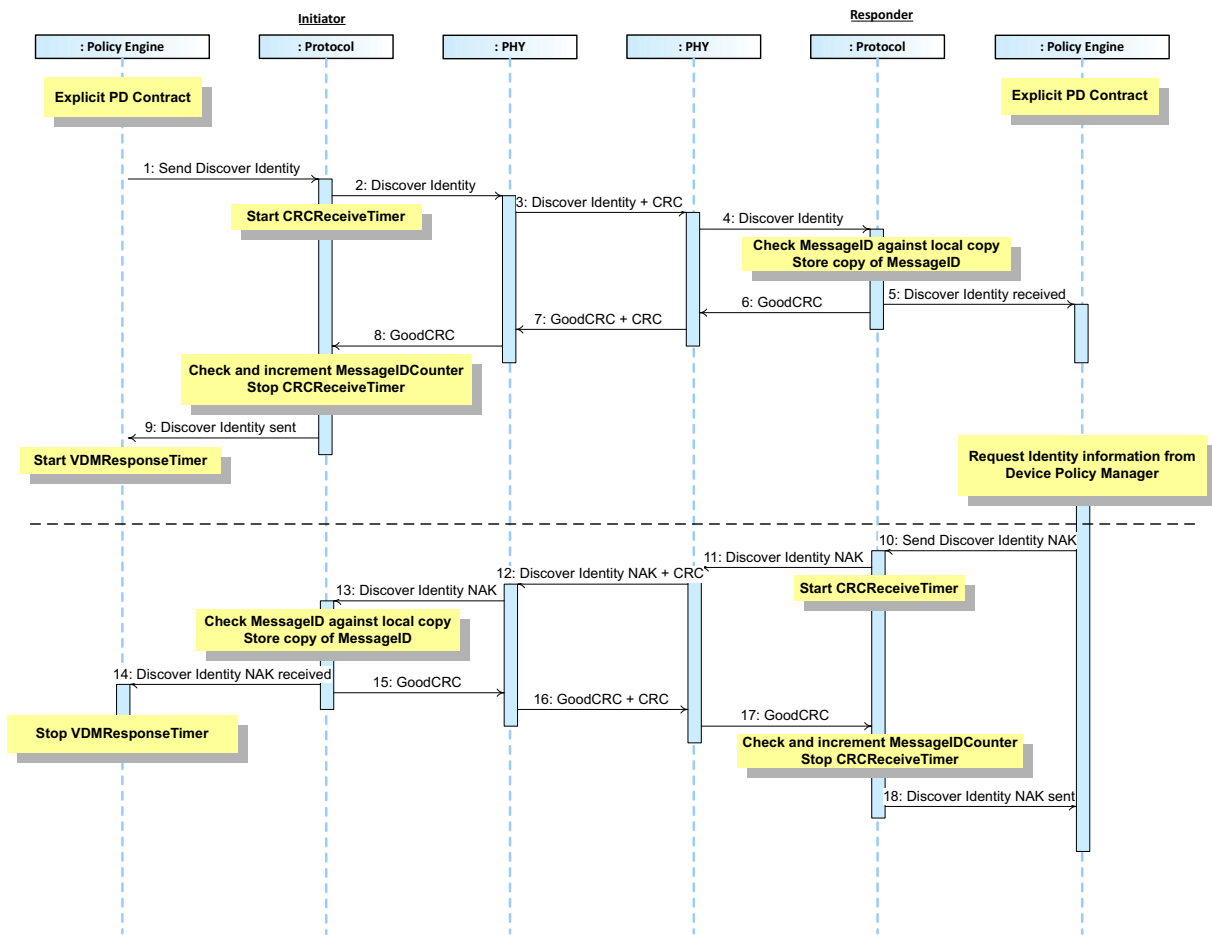
Step	Initiator	Responder
1	The <i>Initiator</i> has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover Identity Command request.	The <i>Responder</i> has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover Identity Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover Identity Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover Identity Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover Identity Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Identity Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover Identity Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover Identity Command ACK response.
11		<i>Protocol Layer</i> creates the Discover Identity Command ACK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover Identity Command ACK response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover Identity Command ACK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Identity Command ACK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.129 Steps for Initiator to UFP Discover Identity (ACK)

Step	Initiator	Responder
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Discover Identity Command ACK</i> response was successfully sent.

Figure 8.103, "*Initiator to Responder Discover Identity (NAK)*" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* attempts to discover identity information from the *Responder* but receives a **NAK**.

Figure 8.103 Initiator to Responder Discover Identity (NAK)



[Table 8.130, "Steps for Initiator to UFP Discover Identity \(NAK\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.103, "Initiator to Responder Discover Identity \(NAK\)"](#) above.

Table 8.130 Steps for Initiator to UFP Discover Identity (NAK)

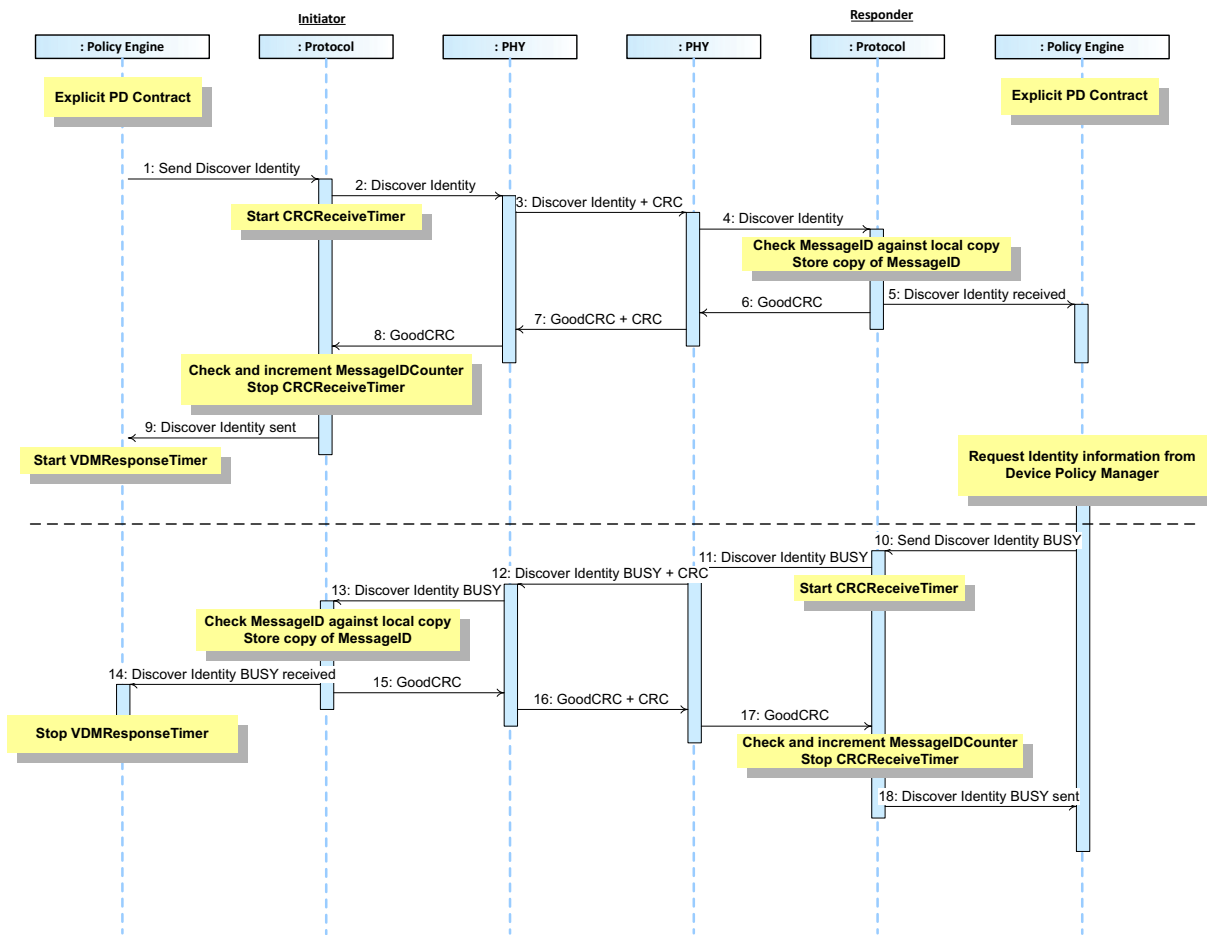
Step	Initiator	Responder
1	The <i>Initiator</i> has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover Identity Command request.	The <i>Responder</i> has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover Identity Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover Identity Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover Identity Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover Identity Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Identity Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover Identity Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover Identity Command NAK response.
11		<i>Protocol Layer</i> creates the Discover Identity Command NAK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover Identity Command NAK response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover Identity Command NAK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Identity Command NAK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.130 Steps for Initiator to UFP Discover Identity (NAK)

Step	Initiator	Responder
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Discover Identity Command NAK response was successfully sent.</i>

Figure 8.104, "Initiator to Responder Discover Identity (BUSY)" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* attempts to discover identity information from the *Responder* but receives a **BUSY**.

Figure 8.104 Initiator to Responder Discover Identity (BUSY)



[Table 8.131, "Steps for Initiator to UFP Discover Identity \(BUSY\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.104, "Initiator to Responder Discover Identity \(BUSY\)"](#) above.

Table 8.131 Steps for Initiator to UFP Discover Identity (BUSY)

Step	Initiator	Responder
1	The <i>Initiator</i> has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover Identity Command request.	The <i>Responder</i> has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover Identity Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover Identity Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover Identity Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover Identity Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Identity Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover Identity Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover Identity Command BUSY response.
11		<i>Protocol Layer</i> creates the Discover Identity Command BUSY response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover Identity Command BUSY response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover Identity Command BUSY response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Identity Command BUSY response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.131 Steps for Initiator to UFP Discover Identity (BUSY)

Step	Initiator	Responder
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Discover Identity Command NAK response was successfully sent.</i>

8.3.2.14.2

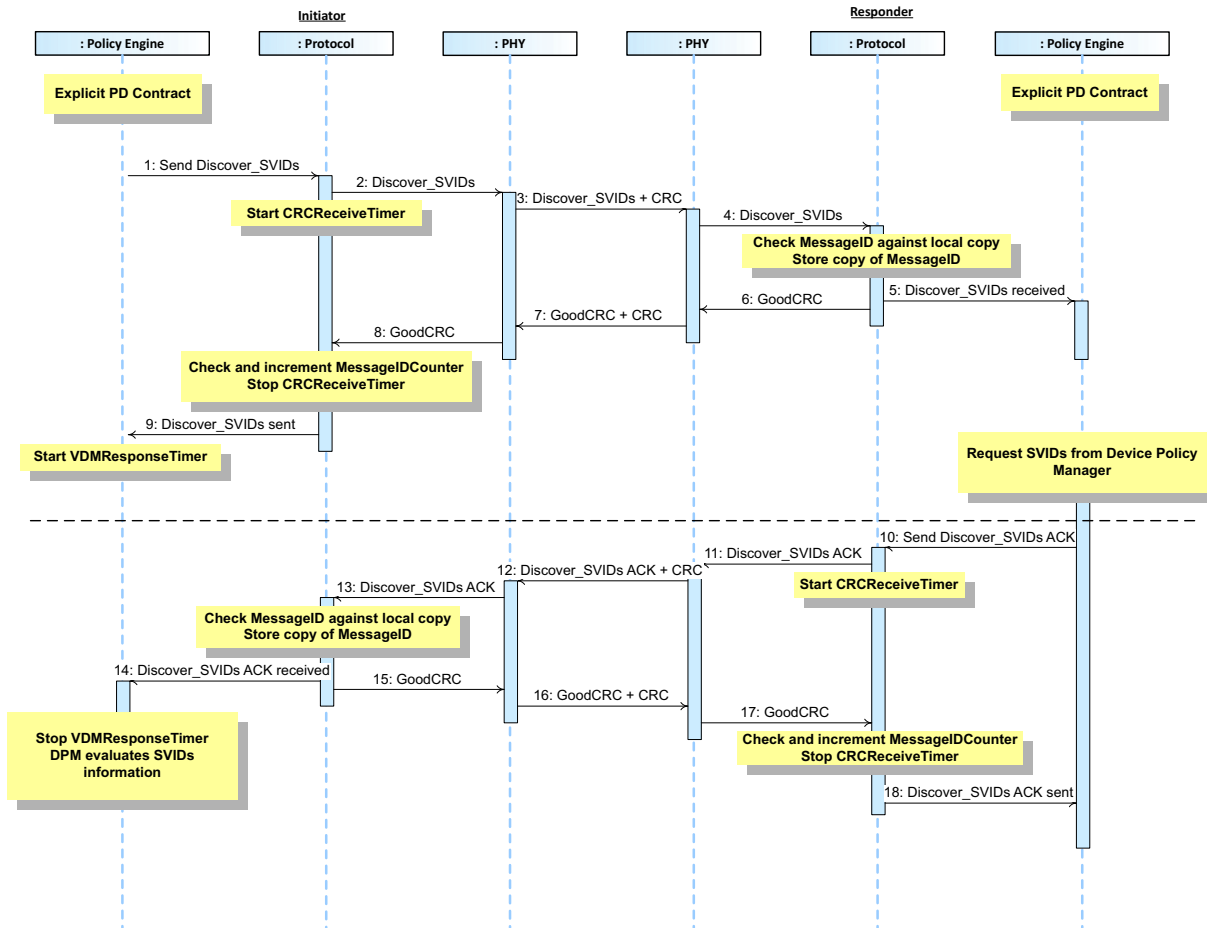
Discover SVIDs

8.3.2.14.2.1

Initiator to Responder Discover SVIDs (ACK)

Figure 8.105, "Initiator to Responder Discover SVIDs (ACK)" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* discovers *SVID* information from the *Responder*.

Figure 8.105 Initiator to Responder Discover SVIDs (ACK)



[Table 8.132, "Steps for DFP to UFP Discover SVIDs \(ACK\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.105, "Initiator to Responder Discover SVIDs \(ACK\)"](#) above.

Table 8.132 Steps for DFP to UFP Discover SVIDs (ACK)

Step	Initiator	Responder
1	The <i>Initiator</i> has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover SVIDs Command request.	The <i>Responder</i> has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover SVIDs Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover SVIDs Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover SVIDs Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover SVIDs Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover SVIDs Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover SVIDs Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover SVIDs Command ACK response.
11		<i>Protocol Layer</i> creates the Discover SVIDs Command ACK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover SVIDs Command ACK response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover SVIDs Command ACK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover SVIDs Command ACK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

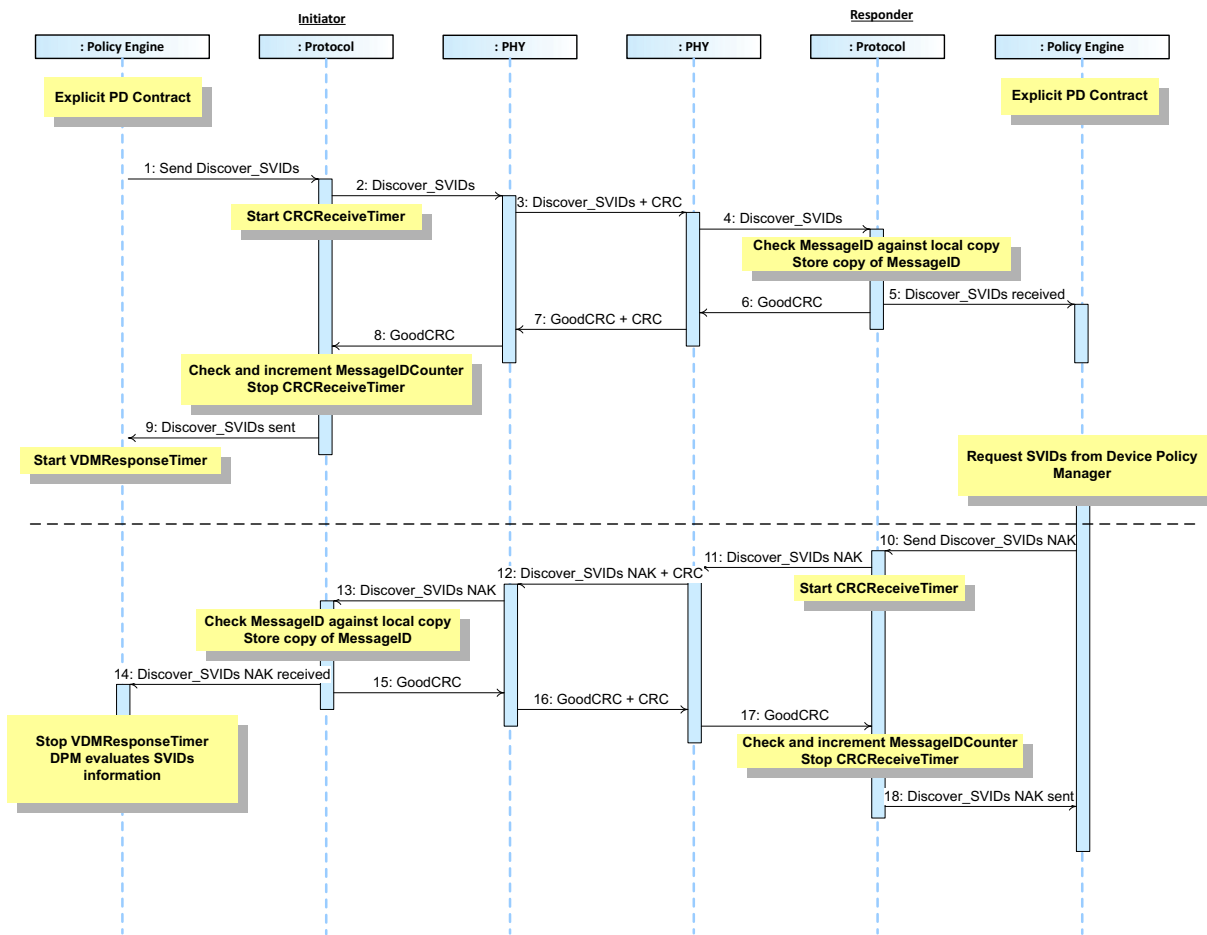
Table 8.132 Steps for DFP to UFP Discover SVIDs (ACK)

Step	Initiator	Responder
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Discover SVIDs Command ACK response was successfully sent.</i>

Initiator to Responder Discover SVIDs (NAK)

Figure 8.106, "Initiator to Responder Discover SVIDs (NAK)" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* attempts to discover *SVID* information from the *Responder* but receives a **NAK**.

Figure 8.106 Initiator to Responder Discover SVIDs (NAK)



[Table 8.133. "Steps for DFP to UFP Discover SVIDs \(NAK\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.106. "Initiator to Responder Discover SVIDs \(NAK\)"](#) above.

Table 8.133 Steps for DFP to UFP Discover SVIDs (NAK)

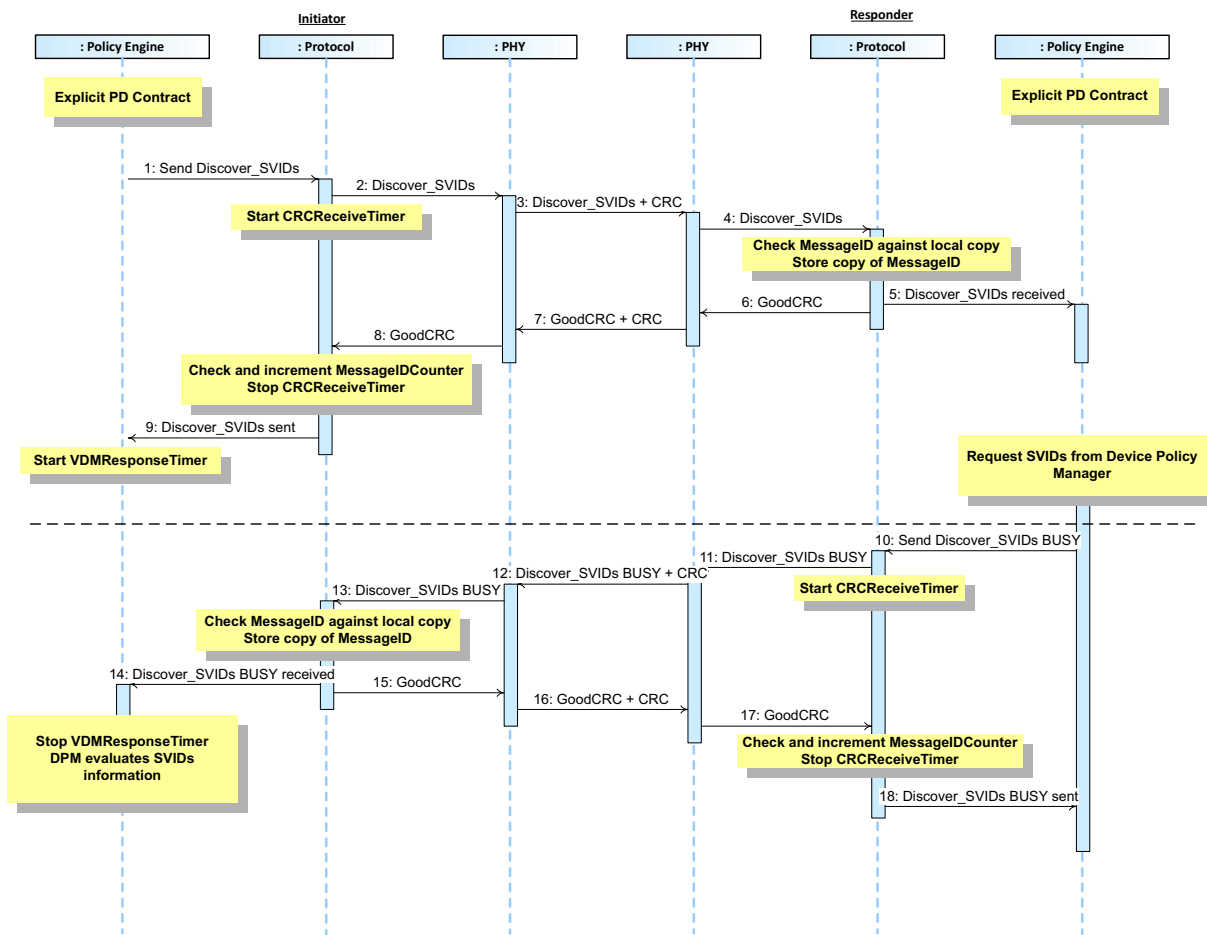
Step	Initiator	Responder
1	The <i>Initiator</i> has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover SVIDs Command request.	The <i>Responder</i> has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover SVIDs Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover SVIDs Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover SVIDs Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover SVIDs Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover SVIDs Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover SVIDs Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover SVIDs Command NAK response.
11		<i>Protocol Layer</i> creates the Discover SVIDs Command NAK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover SVIDs Command NAK response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover SVIDs Command NAK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover SVIDs Command NAK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

Table 8.133 Steps for DFP to UFP Discover SVIDs (NAK)

Step	Initiator	Responder
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Discover SVIDs Command NAK</i> response was successfully sent.

Figure 8.107, "Initiator to Responder Discover SVIDs (BUSY)" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* attempts to discover *SVID* information from the *Responder* but receives a **BUSY**.

Figure 8.107 Initiator to Responder Discover SVIDs (BUSY)



[Table 8.134, "Steps for DFP to UFP Discover SVIDs \(BUSY\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.107, "Initiator to Responder Discover SVIDs \(BUSY\)"](#) above.

Table 8.134 Steps for DFP to UFP Discover SVIDs (BUSY)

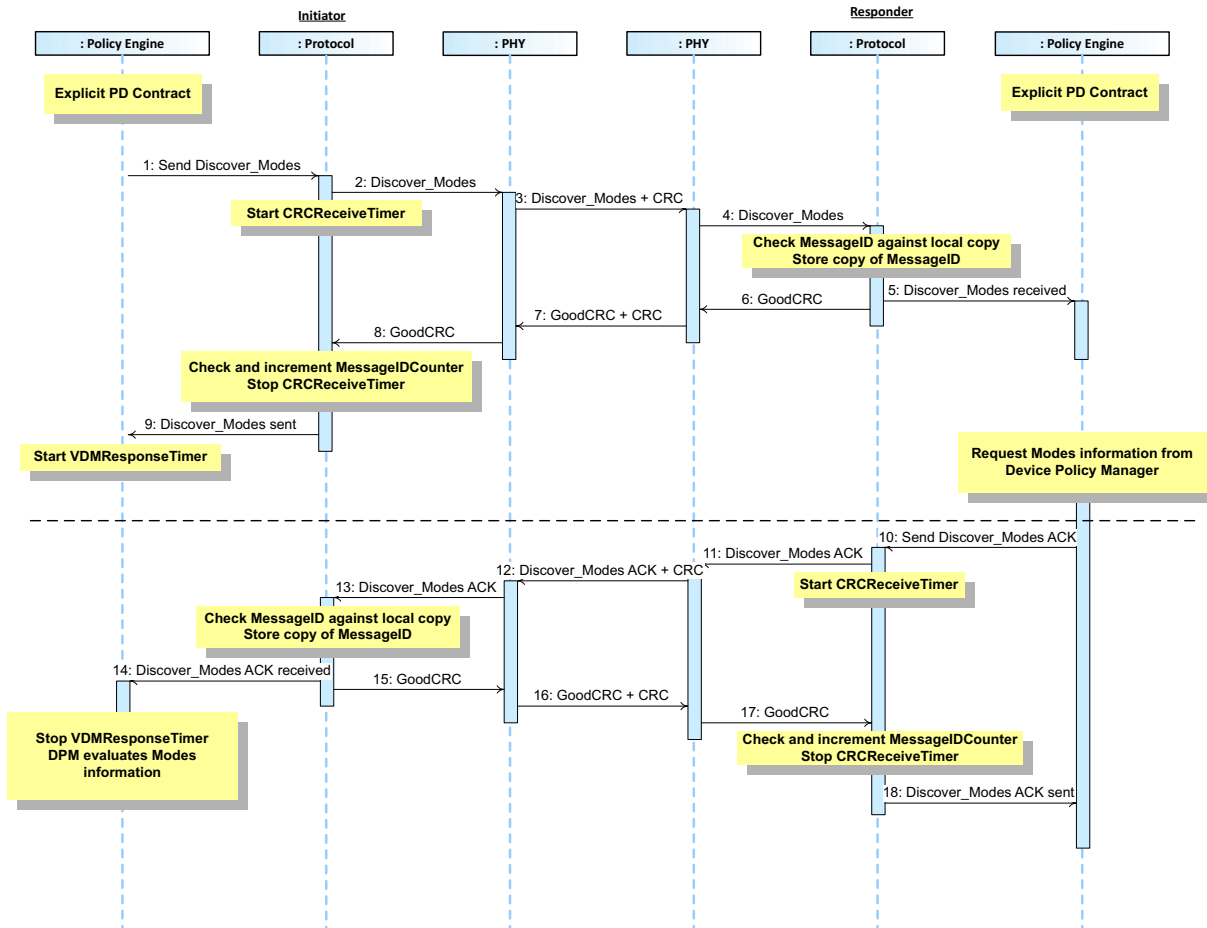
Step	Initiator	Responder
1	The <i>Initiator</i> has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover SVIDs Command request.	The <i>Responder</i> has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover SVIDs Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover SVIDs Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover SVIDs Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover SVIDs Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover SVIDs Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover SVIDs Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover SVIDs Command BUSY response.
11		<i>Protocol Layer</i> creates the Discover SVIDs Command BUSY response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover SVIDs Command BUSY response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover SVIDs Command BUSY response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover SVIDs Command BUSY response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

Table 8.134 Steps for DFP to UFP Discover SVIDs (BUSY)

Step	Initiator	Responder
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Discover SVIDs Command BUSY</i> response was successfully sent.

Figure 8.108, "Initiator to Responder Discover Modes (ACK)" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* discovers Mode information from the *Responder*.

Figure 8.108 Initiator to Responder Discover Modes (ACK)



[Table 8.135, "Steps for DFP to UFP Discover Modes \(ACK\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.108, "Initiator to Responder Discover Modes \(ACK\)"](#).

Table 8.135 Steps for DFP to UFP Discover Modes (ACK)

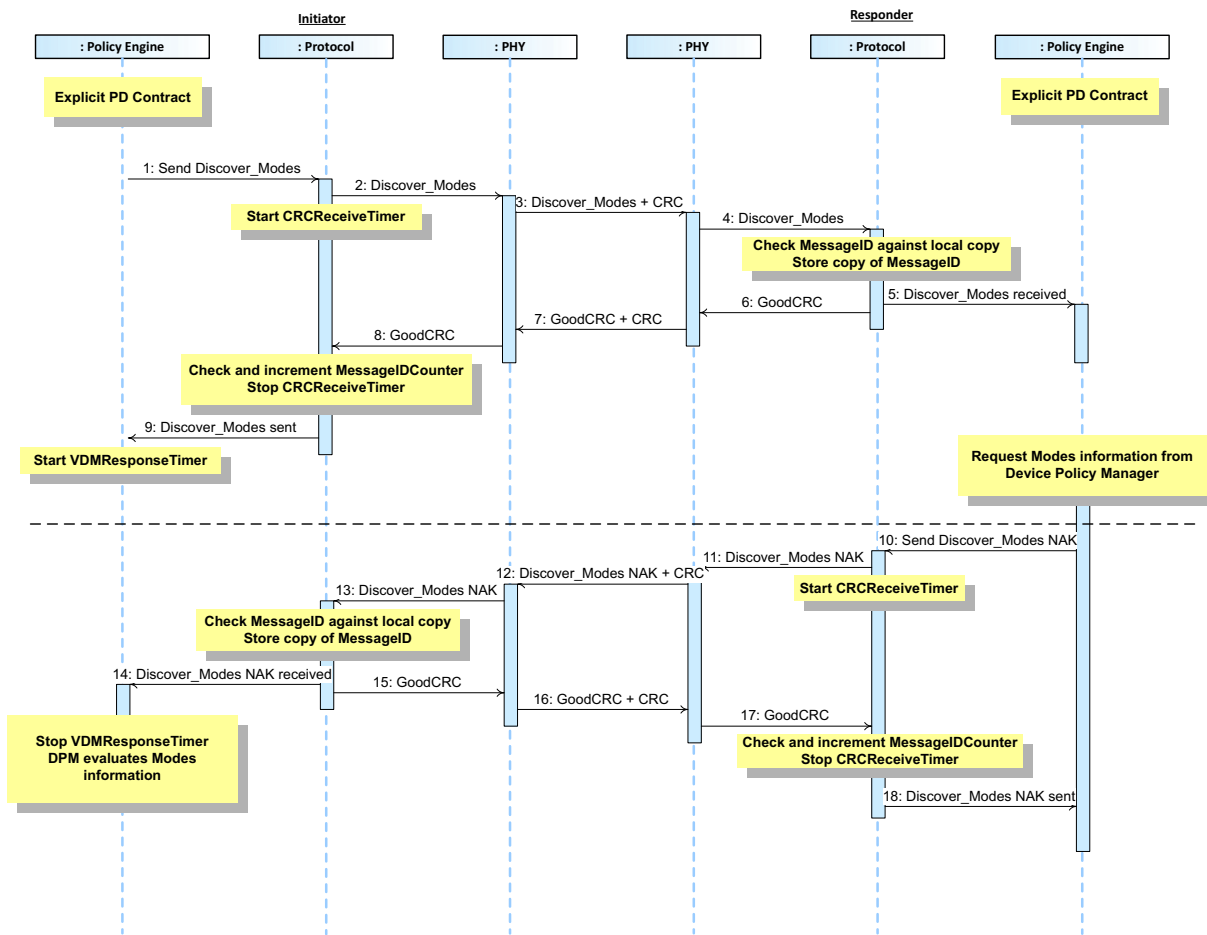
Step	DFP	UFP
1	The DFP has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover Modes Command request.	The UFP has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover Modes Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover Modes Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover Modes Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover Modes Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Modes Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC <i>Message</i> and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover Modes Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover Modes Command ACK response.
11		<i>Protocol Layer</i> creates the Discover Modes Command ACK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover Modes Command ACK response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover Modes Command ACK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Modes Command ACK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .	

Table 8.135 Steps for DFP to UFP Discover Modes (ACK)

Step	DFP	UFP
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Discover Modes Command ACK response was successfully sent.</i>

Figure 8.109, "Initiator to Responder Discover Modes (NAK)" shows an example sequence between an *Initiator* and *Responder*, where both *Port Partners* are in an *Explicit Contract* and the *Initiator* attempts to discover Mode information from the *Responder* but receives a **NAK**.

Figure 8.109 Initiator to Responder Discover Modes (NAK)



[Table 8.136, "Steps for DFP to UFP Discover Modes \(NAK\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.109, "Initiator to Responder Discover Modes \(NAK\)"](#).

Table 8.136 Steps for DFP to UFP Discover Modes (NAK)

Step	DFP	UFP
1	The DFP has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover Modes Command request.	The UFP has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover Modes Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Discover Modes Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover Modes Command request and checks the CRC to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the CRC and forwards the Discover Modes Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Modes Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC <i>Message</i> and checks the CRC to verify the <i>Message</i> .	<i>PHY Layer</i> appends CRC and sends the GoodCRC <i>Message</i> .
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover Modes Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the DPM. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover Modes Command NAK response.
11		<i>Protocol Layer</i> creates the Discover Modes Command NAK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover Modes Command NAK response and compares the CRC it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a CRC and sends the Discover Modes Command NAK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Modes Command NAK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the DPM for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .	

Table 8.136 Steps for DFP to UFP Discover Modes (NAK)

Step	DFP	UFP
16	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
17		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
18		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Discover Modes Command NAK response was successfully sent.</i>

Initiator to Responder Discover Modes (BUSY)

[Table 8.137, "Steps for DFP to UFP Discover Modes \(BUSY\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.110, "Initiator to Responder Discover Modes \(BUSY\)"](#).

Table 8.137 Steps for DFP to UFP Discover Modes (BUSY)

Step	DFP	UFP
1	The DFP has an <i>Explicit Contract</i> . The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Discover Modes Command request.	The UFP has an <i>Explicit Contract</i> .
2	<i>Protocol Layer</i> creates the Discover Modes Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Discover Modes Command request. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Discover Modes Command request and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Discover Modes Command request to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Modes Command request information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC <i>Message</i> and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC <i>Message</i> .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC <i>Message</i> to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Discover Modes Command request was successfully sent. <i>Policy Engine</i> starts the VDMResponseTimer .	
10		<i>Policy Engine</i> requests the identity information from the <i>DPM</i> . The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Discover Modes Command NAK response.
11		<i>Protocol Layer</i> creates the Discover Modes Command NAK response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Discover Modes Command NAK response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Discover Modes Command NAK response. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Discover Modes Command NAK response information to the <i>Policy Engine</i> that consumes it.	
14	The <i>Policy Engine</i> stops the VDMResponseTimer and passed the Identity information to the <i>DPM</i> for evaluation.	
15	<i>Protocol Layer</i> generates a GoodCRC <i>Message</i> and passes it <i>PHY Layer</i> .	

Table 8.137 Steps for DFP to UFP Discover Modes (BUSY)

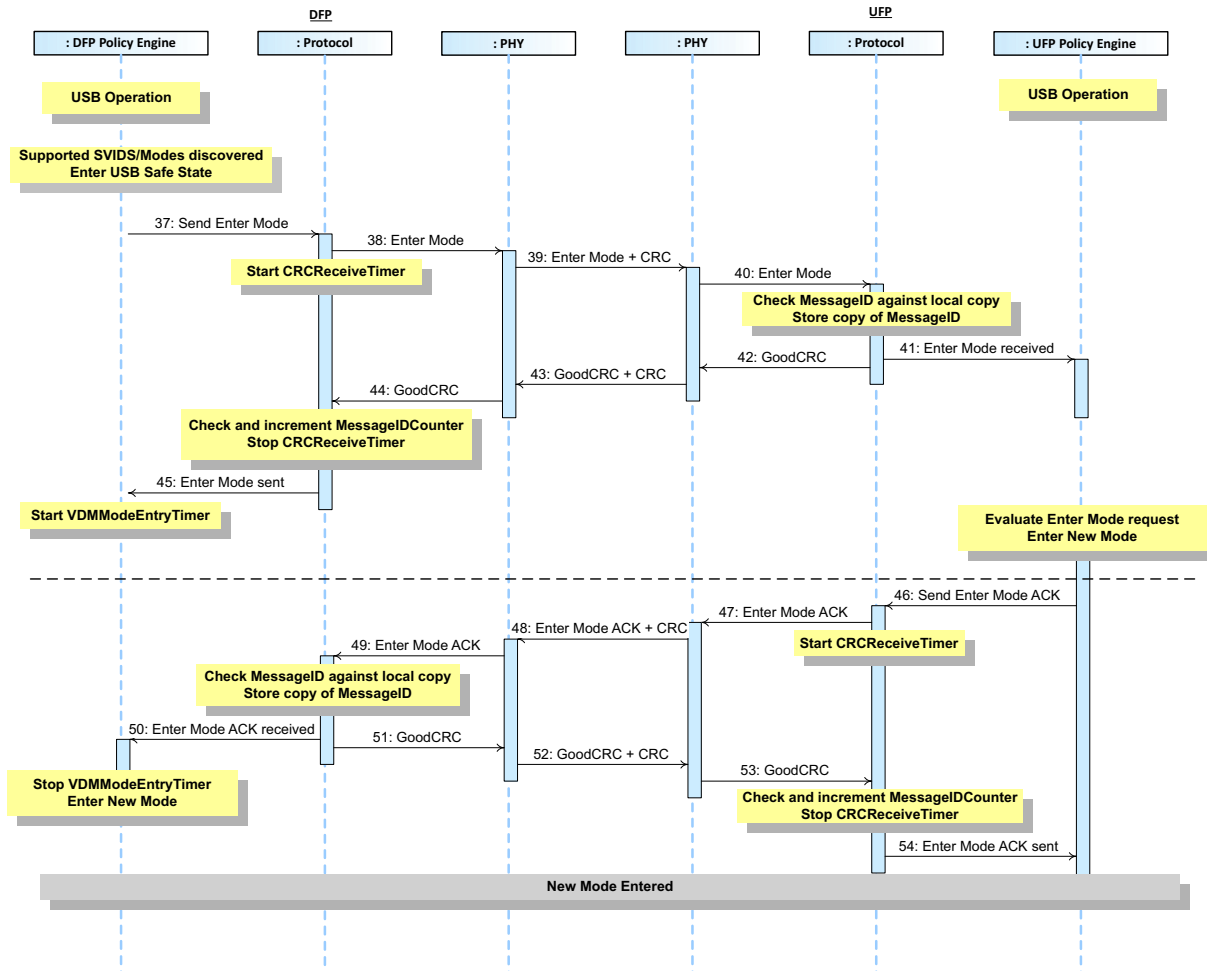
Step	DFP	UFP
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Discover Modes Command NAK</i> response was successfully sent.

Enter/Exit Mode

DFP to UFP Enter Mode

Figure 8.111, "DFP to UFP Enter Mode" shows an example sequence between a DFP and a UFP that occurs after the DFP has discovered supported SVIDs and Modes at which point it selects and enters a Mode.

Figure 8.111 DFP to UFP Enter Mode



[Table 8.138, "Steps for DFP to UFP Enter Mode"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.111, "DFP to UFP Enter Mode"](#) above.

Table 8.138 Steps for DFP to UFP Enter Mode

Step	DFP	UFP
1	<p>The <i>DFP</i> has an <i>Explicit Contract</i></p> <p>The <i>DFP</i> has discovered the supported <i>SVIDS</i> using the <i>Discover SVIDs Command</i> request and the supported Modes using the <i>Discover Modes Command</i> request</p> <p>The <i>DFP</i> goes to <i>USB Safe State</i>. The <i>DPM</i> requests the <i>Policy Engine</i> to enter a Mode.</p> <p>The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send an <i>Enter Mode Command</i> request.</p>	<p>The <i>UFP</i> has an <i>Explicit Contract</i>.</p>
2	<p><i>Protocol Layer</i> creates the <i>Enter Mode Command</i> request and passes to <i>PHY Layer</i>.</p>	
3	<p><i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Enter Mode Command</i> request. Starts <i>CRCReceiveTimer</i>.</p>	<p><i>PHY Layer</i> receives the <i>Enter Mode Command</i> request and checks the <i>CRC</i> to verify the <i>Message</i>.</p>
4		<p><i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Enter Mode Command</i> request to the <i>Protocol Layer</i>.</p>
5		<p><i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received <i>Enter Mode Command</i> request information to the <i>Policy Engine</i> that consumes it.</p>
6		<p><i>Protocol Layer</i> generates a <i>GoodCRC Message</i> and passes it <i>PHY Layer</i>.</p>
7	<p><i>PHY Layer</i> receives the <i>GoodCRC Message</i> and checks the <i>CRC</i> to verify the <i>Message</i>.</p>	<p><i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC Message</i>.</p>
8	<p><i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i>.</p>	
9	<p><i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i>.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter Mode Command</i> request was successfully sent.</p> <p><i>Policy Engine</i> starts the <i>VDMModeEntryTimer</i>.</p>	
10		<p><i>Policy Engine</i> requests the <i>DPM</i> to enter the new Mode. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Enter Mode Command ACK</i> response.</p>
11		<p><i>Protocol Layer</i> creates the <i>Enter Mode Command ACK</i> response and passes to <i>PHY Layer</i>.</p>
12	<p><i>PHY Layer</i> receives the <i>Enter Mode Command ACK</i> response and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i>.</p>	<p><i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Enter Mode Command ACK</i> response. Starts <i>CRCReceiveTimer</i>.</p>
13	<p><i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received <i>Enter Mode Command ACK</i> response information to the <i>Policy Engine</i> that consumes it.</p>	
14	<p>The <i>Policy Engine</i> stops the <i>VDMModeEntryTimer</i> and requests the <i>DPM</i> to enter the new Mode.</p>	

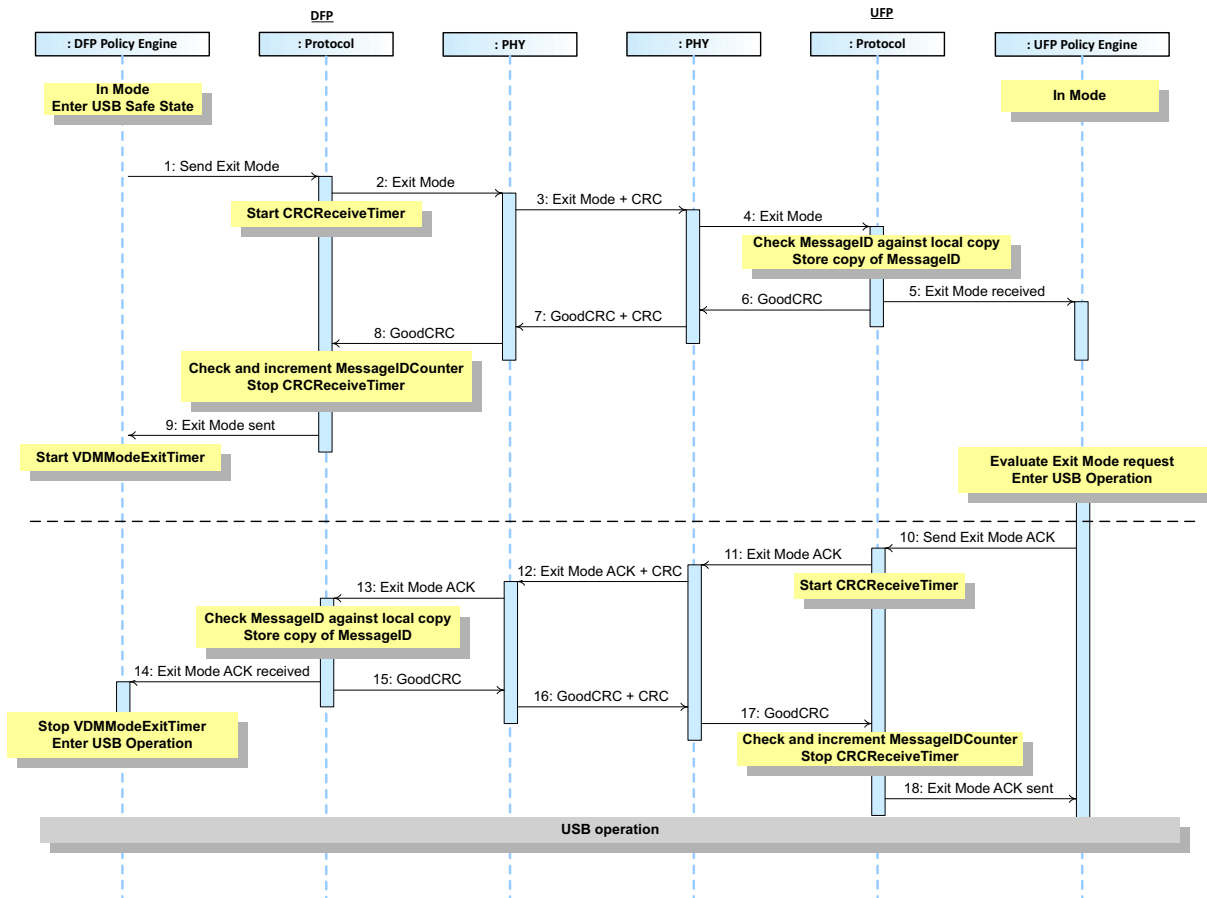
Table 8.138 Steps for DFP to UFP Enter Mode

Step	DFP	UFP
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Enter Mode Command ACK response was successfully sent.
DFP and UFP are operating in the new Mode		

DFP to UFP Exit Mode

Figure 8.112, "DFP to UFP Exit Mode" shows an example sequence between a DFP and a UFP, where the DFP commands the UFP to exit the only Active Mode.

Figure 8.112 DFP to UFP Exit Mode



[Table 8.139, "Steps for DFP to UFP Exit Mode"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.112, "DFP to UFP Exit Mode"](#) above.

Table 8.139 Steps for DFP to UFP Exit Mode

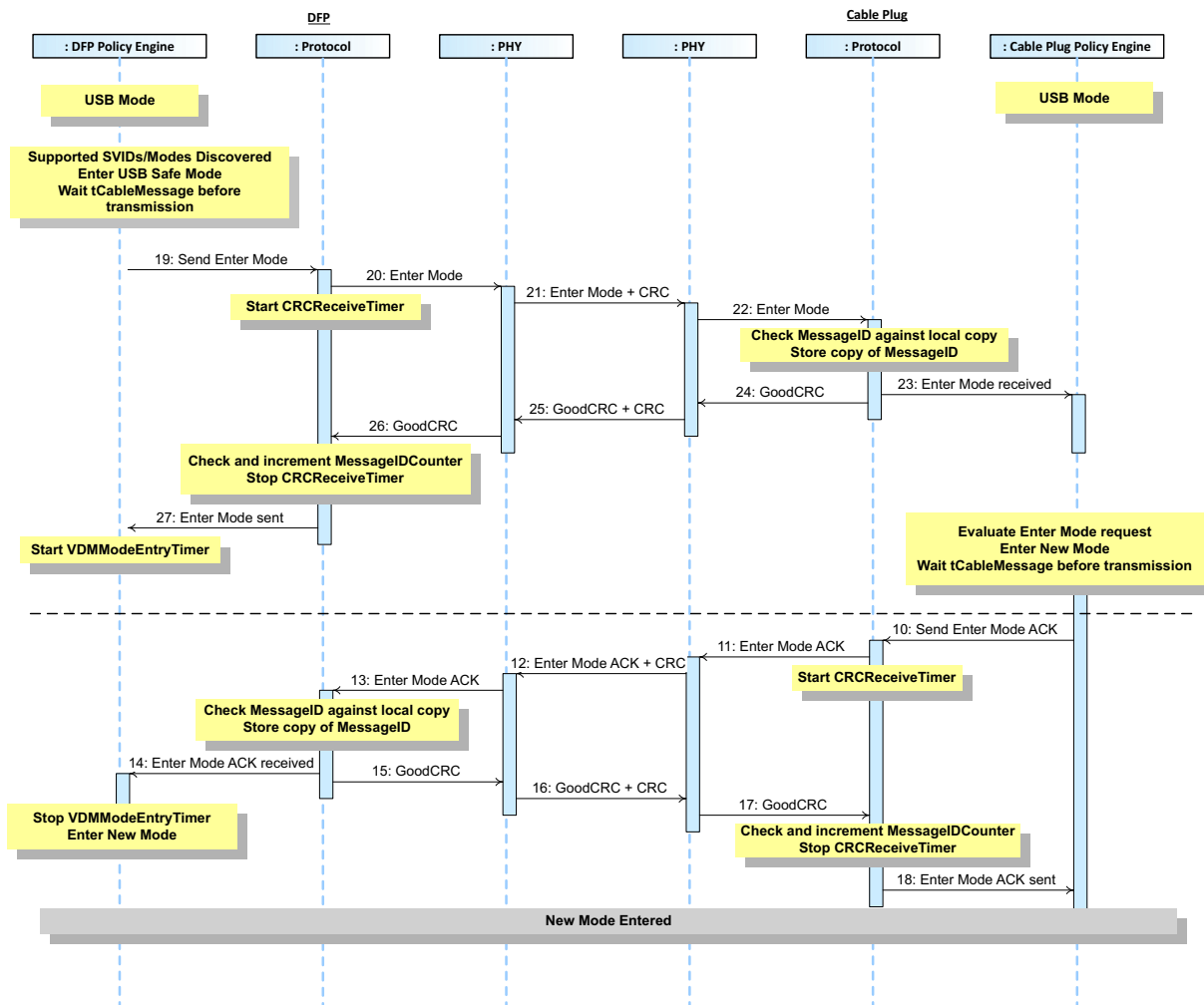
Step	DFP	UFP
1	The DFP is in a Mode and then enters <i>USB Safe State</i> . The Policy Engine directs the Protocol Layer to send an Exit Mode Command request.	The UFP is in a Mode.
2	Protocol Layer creates the Exit Mode Command request and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Exit Mode Command request. Starts CRCReceiveTimer .	PHY Layer receives the Exit Mode Command request and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Exit Mode Command request to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Exit Mode Command request information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Exit Mode Command request was successfully sent. Policy Engine starts the VDMModeExitTimer .	
10		Policy Engine requests the DPM to enter USB operation. The Policy Engine tells the Protocol Layer to form an Exit Mode Command ACK response.
11		Protocol Layer creates the Exit Mode Command ACK response and passes to PHY Layer.
12	PHY Layer receives the Exit Mode Command ACK response and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the Exit Mode Command ACK response. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Exit Mode Command ACK response information to the Policy Engine that consumes it.	
14	The Policy Engine stops the VDMModeExitTimer and requests the DPM to enter USB Operation.	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	
16	PHY Layer appends a CRC and sends the GoodCRC Message.	PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.

Table 8.139 Steps for DFP to UFP Exit Mode

Step	DFP	UFP
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Exit Mode Command ACK</i> response was successfully sent.
Both <i>DFP</i> and <i>UFP</i> are in USB Operation		

Figure 8.113, "DFP to Cable Plug Enter Mode" shows an example sequence between a DFP and a Cable Plug that occurs after the DFP has discovered supported SVIDs and Modes at which point it selects and enters a Mode.

Figure 8.113 DFP to Cable Plug Enter Mode



[Table 8.140, "Steps for DFP to Cable Plug Enter Mode"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.113, "DFP to Cable Plug Enter Mode"](#) above.

Table 8.140 Steps for DFP to Cable Plug Enter Mode

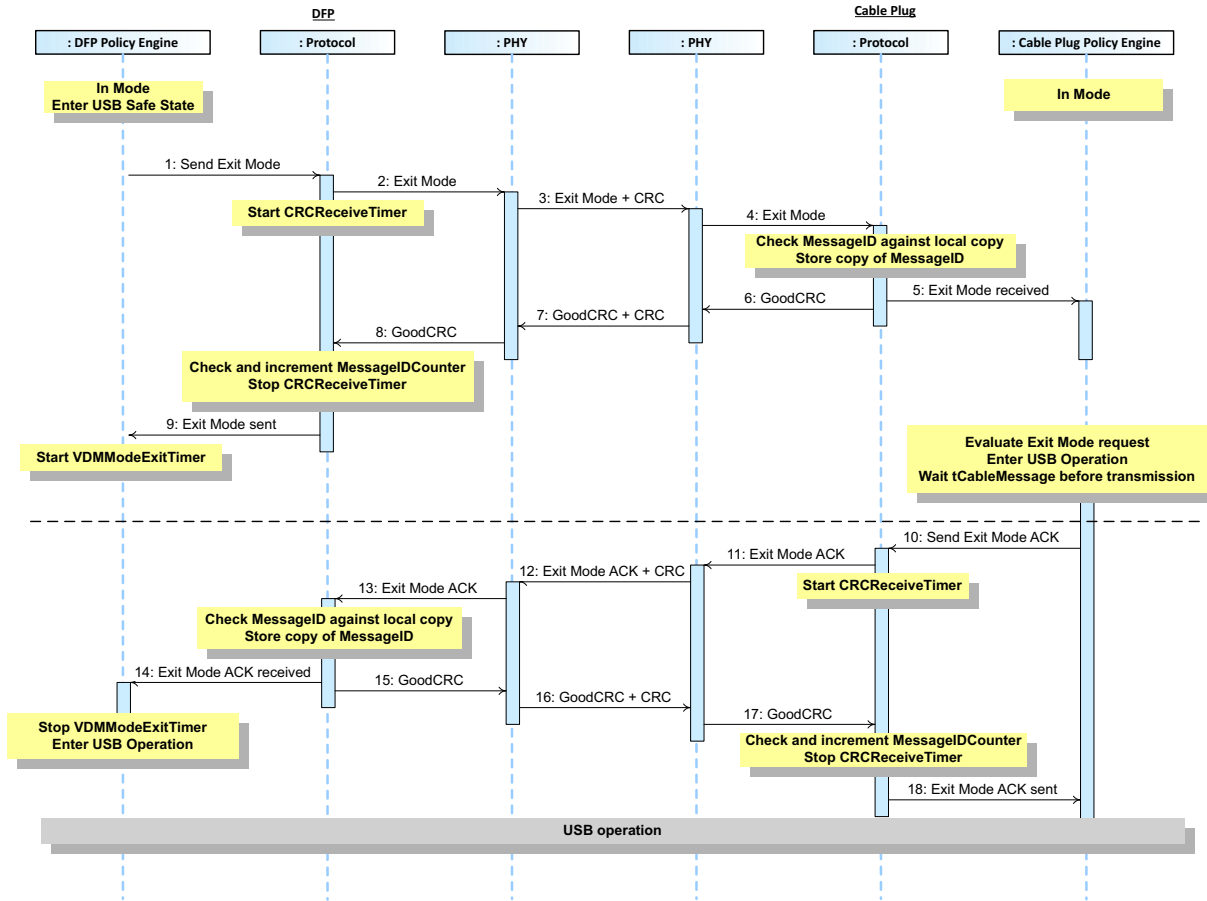
Step	DFP	Cable Plug
1	<p>The DFP has an <i>Explicit Contract</i></p> <p>The DFP has discovered the supported SVIDS using the <i>Discover SVIDs</i> Command request and the supported Modes using the <i>Discover Modes</i> Command request</p> <p>The DFP goes to <i>USB Safe State</i>. The DPM requests the <i>Policy Engine</i> to enter a Mode.</p> <p><i>tCableMessage</i> after the last <i>GoodCRC</i> Message was sent the <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send an <i>Enter Mode</i> Command request.</p>	
2	<i>Protocol Layer</i> creates the <i>Enter Mode</i> Command request and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>Enter Mode</i> Command request. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Enter Mode</i> Command request and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>Enter Mode</i> Command request to the <i>Protocol Layer</i> .
5		<p><i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received <i>Enter Mode</i> Command request information to the <i>Policy Engine</i> that consumes it.</p>
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<p><i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i>.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter Mode</i> Command request was successfully sent.</p> <p><i>Policy Engine</i> starts the <i>VDMModeEntryTimer</i>.</p>	
10		<i>Policy Engine</i> requests the <i>DPM</i> to enter the new Mode. <i>tCableMessage</i> after the <i>GoodCRC</i> Message was sent the <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Enter Mode</i> Command <i>ACK</i> response.
11		<i>Protocol Layer</i> creates the <i>Enter Mode</i> Command <i>ACK</i> response and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the <i>Enter Mode</i> Command <i>ACK</i> response and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>Enter Mode</i> Command <i>ACK</i> response. Starts <i>CRCReceiveTimer</i> .
13	<p><i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received <i>Enter Mode</i> Command <i>ACK</i> response information to the <i>Policy Engine</i> that consumes it.</p>	

Table 8.140 Steps for DFP to Cable Plug Enter Mode

Step	DFP	Cable Plug
14	The <i>Policy Engine</i> stops the <i>VDMModeEntryTimer</i> and requests the <i>DPM</i> to enter the new Mode.	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter Mode</i> Command <i>ACK</i> response was successfully sent.
DFP and Cable Plug are operating in the new Mode		

Figure 8.114, "DFP to Cable Plug Exit Mode" shows an example sequence between a USB Type-C[®] DFP and a Cable Plug, where the DFP commands the Cable Plug to exit an Active Mode.

Figure 8.114 DFP to Cable Plug Exit Mode



[Table 8.141, "Steps for DFP to Cable Plug Exit Mode"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.114, "DFP to Cable Plug Exit Mode"](#) above.

Table 8.141 Steps for DFP to Cable Plug Exit Mode

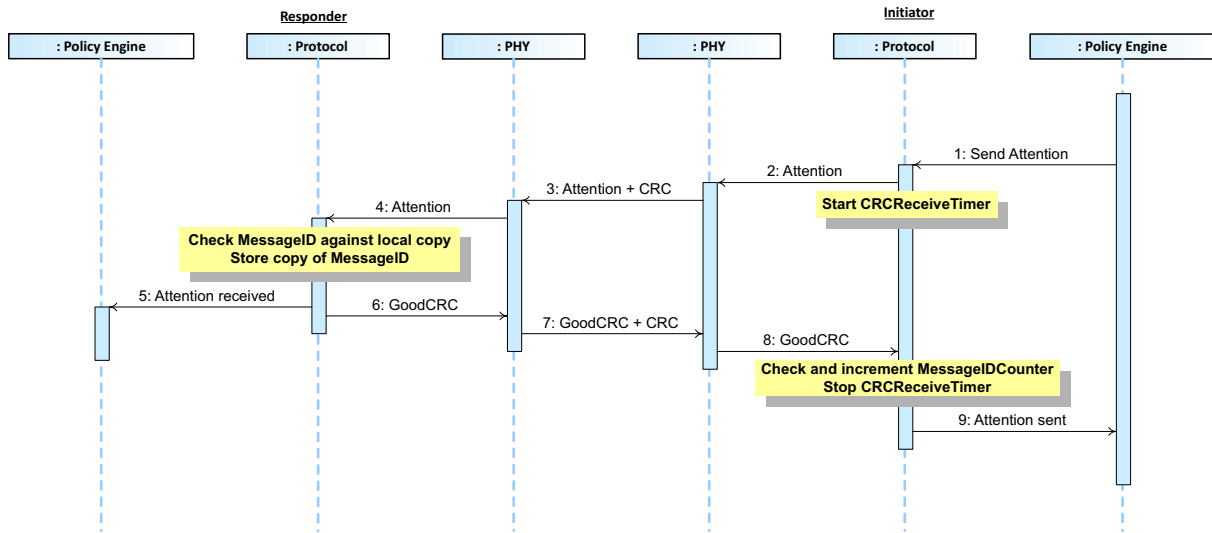
Step	DFP	Cable Plug
1	The DFP is in a Mode and then enters <i>USB Safe State</i> . The Policy Engine directs the Protocol Layer to send an Exit Mode Command request.	The Cable Plug is in a Mode.
2	Protocol Layer creates the Exit Mode Command request and passes to PHY Layer.	
3	PHY Layer appends CRC and sends the Exit Mode Command request. Starts CRCReceiveTimer .	PHY Layer receives the Exit Mode Command request and checks the CRC to verify the Message.
4		PHY Layer removes the CRC and forwards the Exit Mode Command request to the Protocol Layer.
5		Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Exit Mode Command request information to the Policy Engine that consumes it.
6		Protocol Layer generates a GoodCRC Message and passes it PHY Layer.
7	PHY Layer receives the GoodCRC Message and checks the CRC to verify the Message.	PHY Layer appends CRC and sends the GoodCRC Message.
8	PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.	
9	Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer . Protocol Layer informs the Policy Engine that the Exit Mode Command request was successfully sent. Policy Engine starts the VDMModeExitTimer .	
10		Policy Engine requests the DPM to enter USB operation. tCableMessage after the GoodCRC Message was sent the Policy Engine tells the Protocol Layer to form an Exit Mode Command ACK response.
11		Protocol Layer creates the Exit Mode Command ACK response and passes to PHY Layer.
12	PHY Layer receives the Exit Mode Command ACK response and compares the CRC it calculated with the one sent to verify the Message.	PHY Layer appends a CRC and sends the Exit Mode Command ACK response. Starts CRCReceiveTimer .
13	Protocol Layer checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The Protocol Layer forwards the received Exit Mode Command ACK response information to the Policy Engine that consumes it.	
14	The Policy Engine stops the VDMModeExitTimer and requests the DPM to enter USB Operation.	
15	Protocol Layer generates a GoodCRC Message and passes it PHY Layer.	

Table 8.141 Steps for DFP to Cable Plug Exit Mode

Step	DFP	Cable Plug
16	<i>PHY Layer</i> appends a <i>CRC</i> and sends the <i>GoodCRC Message</i> .	<i>PHY Layer</i> receives <i>GoodCRC Message</i> and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
17		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Exit Mode Command ACK</i> response was successfully sent.
Both <i>DFP</i> and <i>Cable Plug</i> are in USB Operation		

Figure 8.115, "Initiator to Responder Attention" shows an example sequence between an *Initiator* and a *Responder*, where the *Initiator* requests attention from the *Responder*.

Figure 8.115 Initiator to Responder Attention



[Table 8.142, "Steps for Initiator to Responder Attention"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.115, "Initiator to Responder Attention"](#) above.

Table 8.142 Steps for Initiator to Responder Attention

Step	Responder	Initiator
1		The <i>DPM</i> requests attention. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Attention Command request.
2		<i>Protocol Layer</i> creates the Attention Command request and passes to <i>PHY Layer</i> .
3	<i>PHY Layer</i> receives the Attention Command request and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Attention Command request. Starts CRCReceiveTimer .
4	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Attention Command request information to the <i>Policy Engine</i> that consumes it.	
5	The <i>Policy Engine</i> informs the <i>DPM</i>	
6	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
7	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .
8		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .
9		<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Attention Command request was successfully sent.

8.3.2.15 Built in Self-Test (BIST)

8.3.2.15.1 BIST Carrier Mode

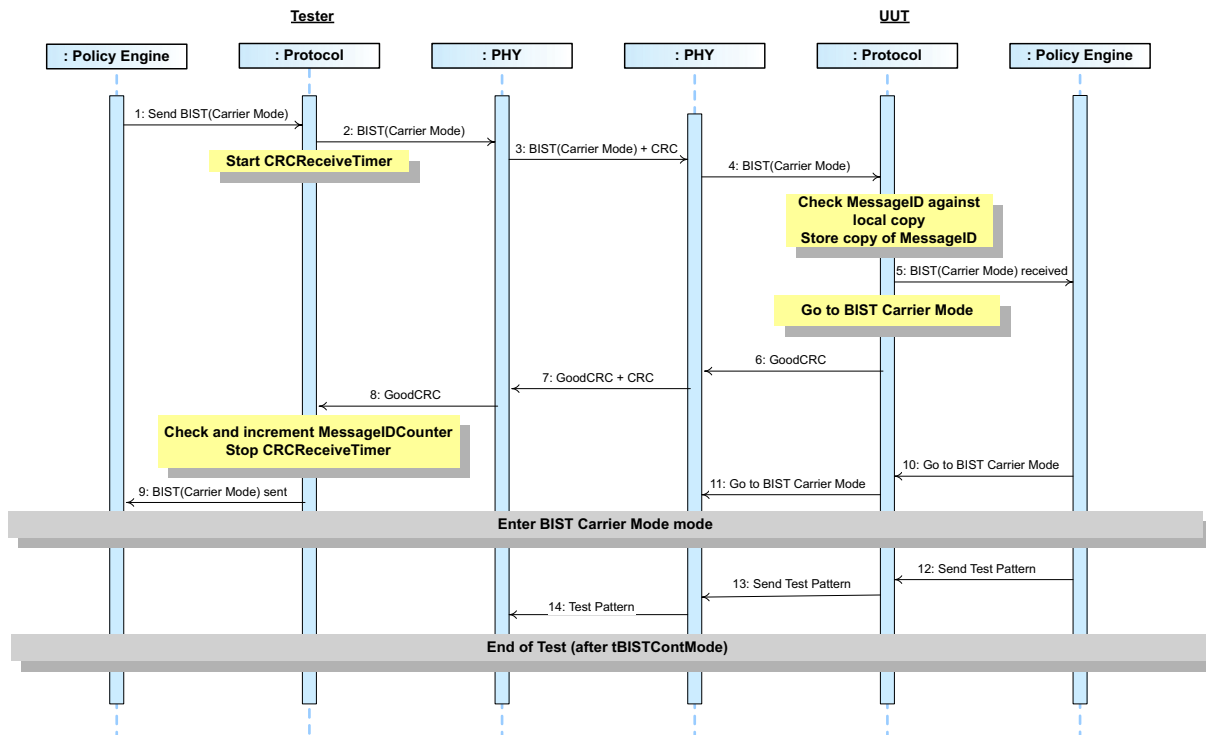
The following is an example of a *BIST Carrier Mode* test between a *Tester* and a *UUT*. When the *UUT* is connected to the *Tester* the sequence below is executed.

[Figure 8.116, "BIST Carrier Mode Test"](#) shows the *Messages* as they flow across the bus and within the devices. This test enables the measurement of power supply noise and frequency drift.

- 1) Connection is established and stable.
- 2) *Tester* sends a **BIST** Message with a **BIST Carrier Mode** BIST Data Object.
- 3) *UUT* answers with a **GoodCRC** Message.
- 4) *UUT* starts sending the *Test Pattern*.
- 5) Operator does the measurements.
- 6) The test ends after **tBISTContMode**.

See also [Section 5.9.1, "BIST Carrier Mode"](#) and [Section 6.4.3.1, "BIST Carrier Mode"](#).

Figure 8.116 BIST Carrier Mode Test



[Table 8.143, "Steps for BIST Carrier Mode Test"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.116, "BIST Carrier Mode Test"](#) above.

Table 8.143 Steps for BIST Carrier Mode Test

Step	Tester	UUT
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a BIST Message, with a <i>BIST Data Object</i> of BIST Carrier Mode , to put the <i>UUT</i> into <i>BIST Carrier Mode</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the BIST Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the BIST Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the BIST Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received BIST Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the BIST Message was successfully sent.	
10		<i>Policy Engine</i> tells <i>Protocol Layer</i> to go into <i>BIST Carrier Mode</i> . The <i>Policy Engine</i> goes to <i>BIST Carrier Mode</i> .
11		<i>Protocol Layer</i> tells <i>PHY Layer</i> to go into <i>BIST Carrier Mode</i> .
	<i>UUT</i> enters <i>BIST Carrier Mode</i> .	
12		The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to start generation of the <i>Test Pattern</i> .
13		<i>Protocol Layer</i> directs the <i>PHY Layer</i> to generate the <i>Test Pattern</i> .
14	<i>PHY Layer</i> receives the <i>Test Pattern</i> stream.	<i>PHY Layer</i> generates a continuous <i>Test Pattern</i> stream.
The <i>UUT</i> exits <i>BIST Carrier Mode</i> after tBISTContMode .		

8.3.2.15.2 BIST Test Data Mode

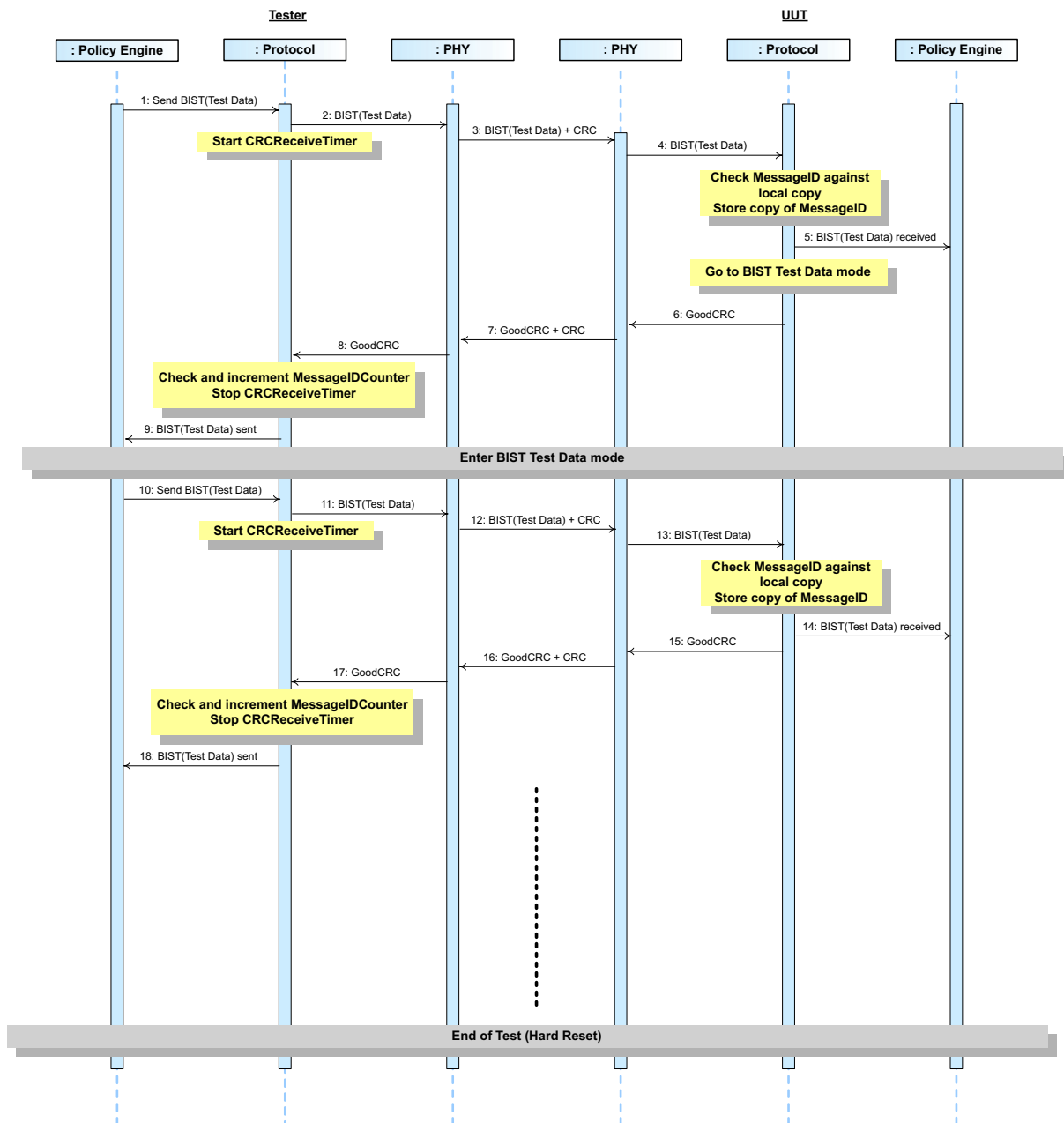
The following is an example of a *BIST Test Data Mode* test between a *Tester* and a *UUT*. When the *UUT* is connected to the *Tester* the sequence below is executed.

[Figure 8.117, "BIST Test Data Test"](#) shows the *Messages* as they flow across the bus and within the devices.

- 1) Connection is established and stable.
- 2) *Tester* sends a **BIST** Message with a **BIST Test Data** BIST Data Object.
- 3) *UUT* answers with a **GoodCRC** Message.
- 4) Steps 2 and 3 are repeated any number of times.
- 5) The test ends after **Hard Reset** Signaling is issued.

See also [Section 5.9.2, "BIST Test Data Mode"](#) and [Section 6.4.3.2, "BIST Test Data Mode"](#).

Figure 8.117 BIST Test Data Test



[Table 8.144, "Steps for BIST Test Data Test"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.117, "BIST Test Data Test"](#) above.

Table 8.144 Steps for BIST Test Data Test

Step	Tester	UUT
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a BIST Message, with a <i>BIST Data Object</i> of BIST Test Data , to put the UUT into <i>BIST Test Data Mode</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the BIST Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the BIST Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the BIST Message to the <i>Protocol Layer</i> .
5		<p><i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received BIST Message information to the <i>Policy Engine</i> that consumes it.</p> <p>The <i>Policy Engine</i> goes into <i>BIST Test Data Mode</i> where it sends no further Messages except for GoodCRC Messages in response to received Messages (see Section 6.4.3.2, "BIST Test Data Mode").</p>
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the BIST Message was successfully sent.	
	UUT enters <i>BIST Test Data Mode</i> .	
10	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a BIST Message, with a <i>BIST Data Object</i> of BIST Test Data , to put the UUT into <i>BIST Test Data Mode</i> .	
11	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
12	<i>PHY Layer</i> appends CRC and sends the BIST Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the BIST Message and checks the CRC to verify the Message.
13		<i>PHY Layer</i> removes the CRC and forwards the BIST Message to the <i>Protocol Layer</i> .

Table 8.144 Steps for BIST Test Data Test

Step	Tester	UUT
14		<p><i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value.</p> <p>The <i>Protocol Layer</i> forwards the received BIST Message information to the <i>Policy Engine</i> that consumes it.</p> <p>The <i>Policy Engine</i> goes into <i>BIST Test Data Mode</i> where it sends no further <i>Messages</i> except for GoodCRC Messages in response to received <i>Messages</i> (see Section 6.4.3.2, "BIST Test Data Mode").</p>
15		<p><i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i>.</p>
16	<p><i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the <i>Message</i>.</p>	<p><i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.</p>
17	<p><i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i>.</p>	
18	<p><i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer.</p> <p><i>Protocol Layer</i> informs the <i>Policy Engine</i> that the BIST Message was successfully sent.</p>	
	Repeat steps 10-18 any number of times	
The UUT exits <i>BIST Test Data Mode</i> after a <i>Hard Reset</i>		

8.3.2.15.3 BIST Shared Capacity Test Mode

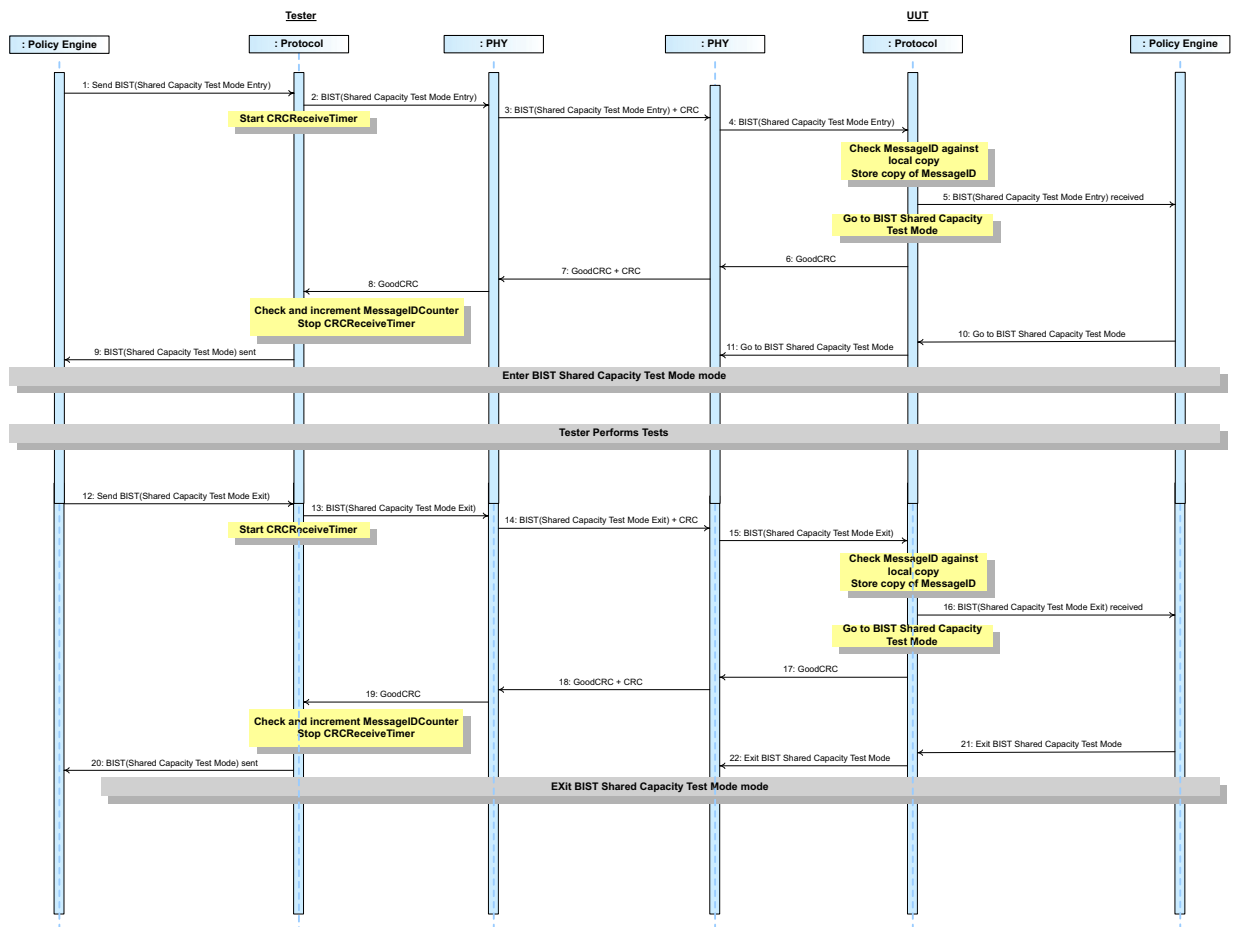
The following is an example of a *BIST Shared Capacity Test Mode* test between a *Tester* and a *UUT*. When the *UUT* is connected to the *Tester* the sequence below is executed.

[Figure 8.118, "BIST Share Capacity Mode Test"](#) shows the *Messages* as they flow across the bus and within the devices. This test places the *UUT* in a compliance test mode where the maximum *Source* capability is always offered on every *Port*, regardless of the availability of shared power i.e., all shared power management is disabled.

- 1) Connection is established and stable.
- 2) *Tester* sends a *BIST* Message with a *BIST Shared Test Mode Entry* *BIST* Data Object.
- 3) *UUT* answers with a *GoodCRC* Message.
- 4) *UUT* enters *BIST Shared Capacity Test Mode*.
- 5) Operator does the measurements.
- 6) *Tester* sends a *BIST* Message with a *BIST Shared Test Mode Exit* *BIST* Data Object.
- 7) *UUT* answers with a *GoodCRC* Message.
- 8) *UUT* exits *BIST Shared Capacity Test Mode*.

See also [Section 5.9.1, "BIST Carrier Mode"](#) and [Section 6.4.3.3, "BIST Shared Capacity Test Mode"](#).

Figure 8.118 BIST Share Capacity Mode Test



[Table 8.145, "Steps for BIST Shared Capacity Test Mode Test"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.118, "BIST Share Capacity Mode Test"](#) above.

Table 8.145 Steps for BIST Shared Capacity Test Mode Test

Step	Tester	UUT
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a BIST Message, with a <i>BIST Data Object</i> of BIST Shared Test Mode Entry , to put the UUT into <i>BIST Shared Capacity Test Mode</i> .	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the BIST Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the BIST Message and checks the CRC to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the BIST Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received BIST Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the BIST Message was successfully sent.	
10		<i>Policy Engine</i> tells <i>Protocol Layer</i> to go into <i>BIST Shared Capacity Test Mode</i> . The <i>Policy Engine</i> goes to <i>BIST Shared Capacity Test Mode</i> .
11		<i>Protocol Layer</i> tells <i>PHY Layer</i> to go into <i>BIST Shared Capacity Test Mode</i> .
UUT enters <i>BIST Shared Capacity Test Mode</i> . Tester performs tests.		
12	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate a BIST Message, with a <i>BIST Data Object</i> of BIST Shared Test Mode Exit , to take the UUT out of <i>BIST Shared Capacity Test Mode</i> .	
13	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
14	<i>PHY Layer</i> appends CRC and sends the BIST Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the BIST Message and checks the CRC to verify the Message.
15		<i>PHY Layer</i> removes the CRC and forwards the BIST Message to the <i>Protocol Layer</i> .
16		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received BIST Message information to the <i>Policy Engine</i> that consumes it.

Table 8.145 Steps for BIST Shared Capacity Test Mode Test

Step	Tester	UUT
17		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
18	<i>PHY Layer</i> receives the GoodCRC and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
19	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
20	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the BIST Message was successfully sent.	
21		<i>Policy Engine</i> tells <i>Protocol Layer</i> to exit <i>BIST Shared Capacity Test Mode</i> . The <i>Policy Engine</i> exits to <i>BIST Shared Capacity Test Mode</i> .
22		<i>Protocol Layer</i> tells <i>PHY Layer</i> to exit <i>BIST Shared Capacity Test Mode</i> .
UUT exits <i>BIST Shared Capacity Test Mode</i> .		

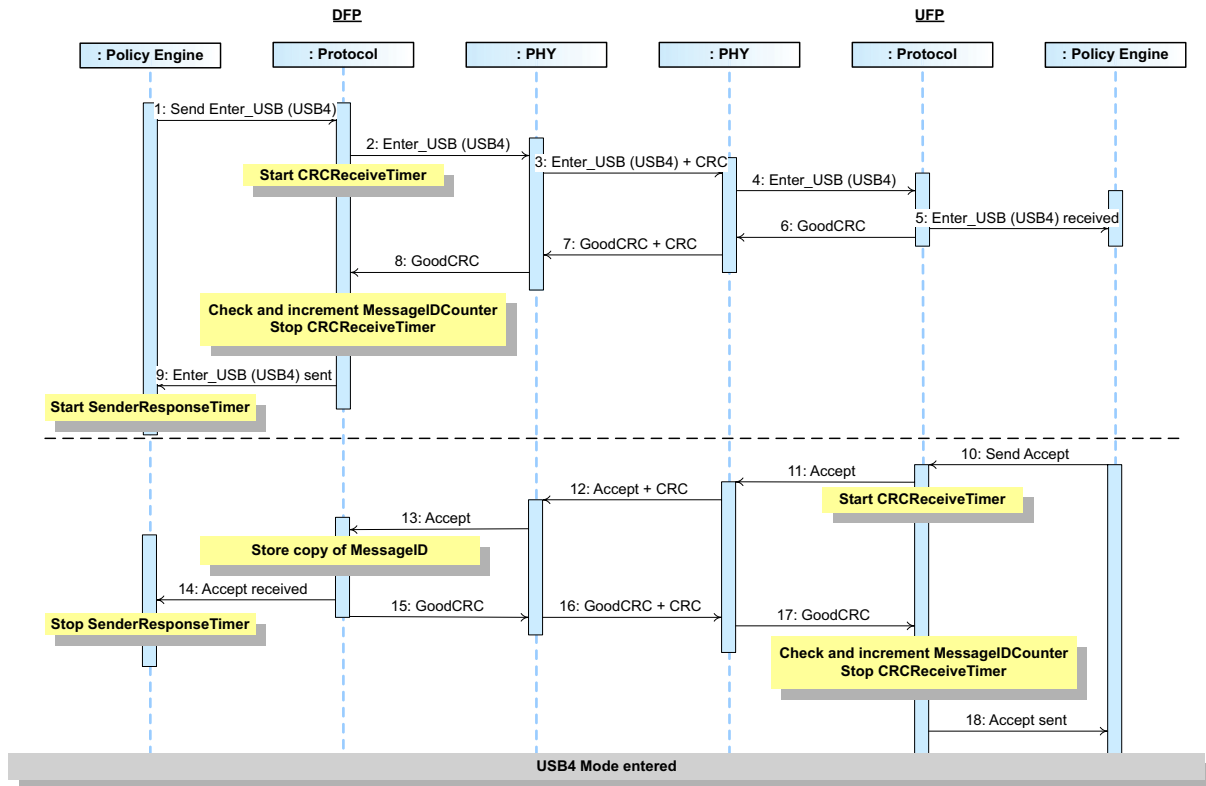
8.3.2.16 Enter USB

8.3.2.16.1 UFP Entering USB4 Mode

8.3.2.16.1.1 UFP Entering USB4 Mode (Accept)

This is an example of an Enter USB operation where the *DFP* requests *[USB4]* mode when this is a **Valid** mode of operation for the *UFP*. [Figure 8.119, "UFP Entering USB4 Mode \(Accept\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter USB process.

Figure 8.119 UFP Entering USB4 Mode (Accept)



[Table 8.146, "Steps for UFP USB4 Mode Entry \(Accept\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.119, "UFP Entering USB4 Mode \(Accept\)"](#) above.

Table 8.146 Steps for UFP USB4 Mode Entry (Accept)

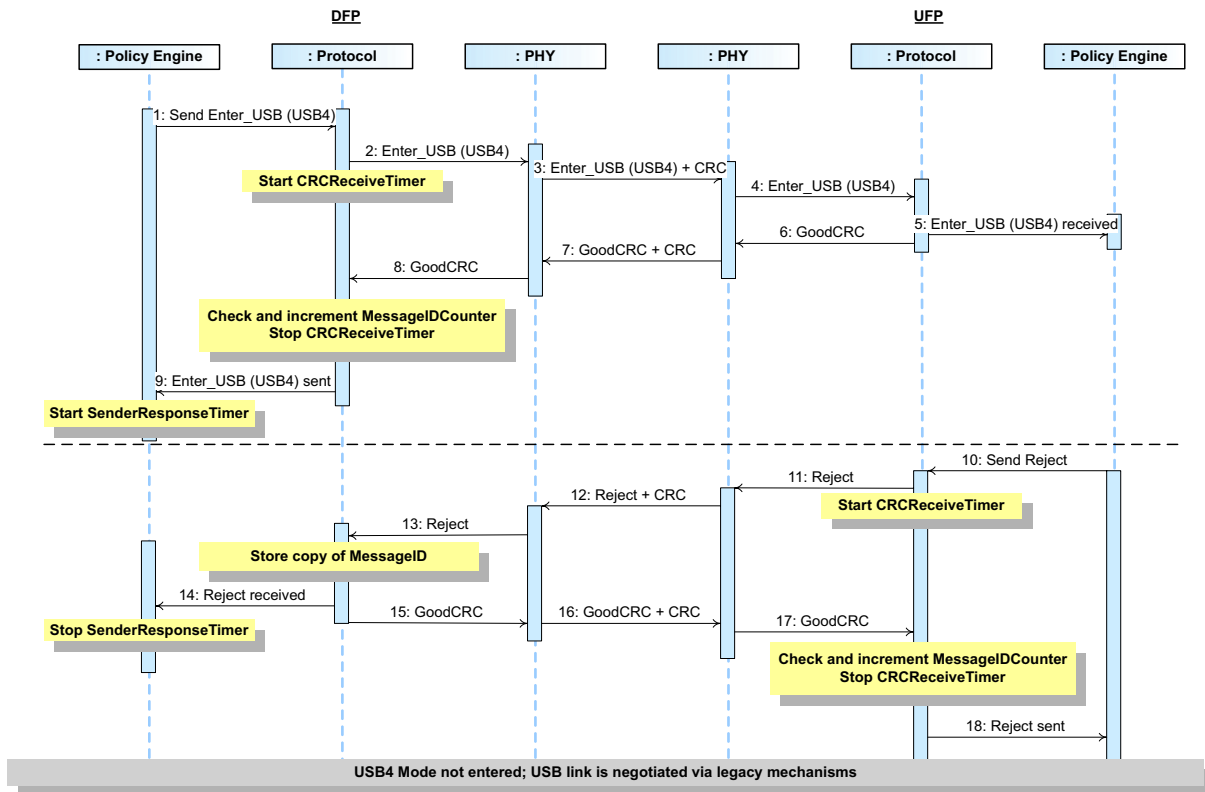
Step	DFP	UFP
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an <i>Enter_USB</i> Message to request entry to <i>[USB4]</i> mode.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the <i>Enter_USB</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Enter_USB</i> Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the <i>Enter_USB</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Enter_USB</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter_USB</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Accept</i> Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received <i>Accept</i> Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .

Table 8.146 Steps for UFP USB4 Mode Entry (Accept)

Step	DFP	UFP
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept Message</i> was successfully sent.
Both <i>Port Partners</i> enter <i>[USB4]</i> operation.		

This is an example of an Enter USB operation where the *DFP* requests *[USB4]* mode when this is an *Invalid* mode of operation for the *UFP*. [Figure 8.120, "UFP Entering USB4 Mode \(Reject\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter USB process.

Figure 8.120 UFP Entering USB4 Mode (Reject)



[Table 8.147, "Steps for UFP USB4 Mode Entry \(Reject\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.120, "UFP Entering USB4 Mode \(Reject\)"](#) above.

Table 8.147 Steps for UFP USB4 Mode Entry (Reject)

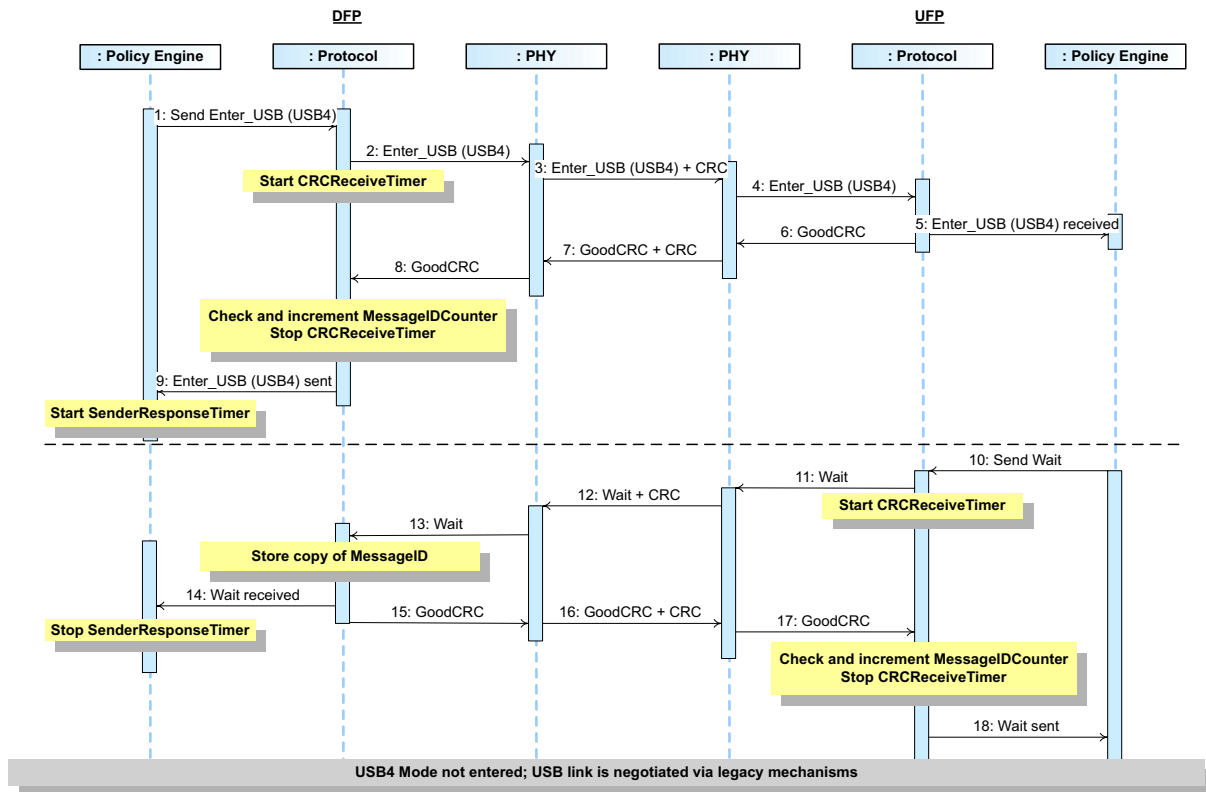
Step	DFP	UFP
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an <i>Enter_USB</i> Message to request entry to <i>[USB4]</i> mode.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the <i>Enter_USB</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Enter_USB</i> Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the <i>Enter_USB</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Enter_USB</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter_USB</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Reject</i> Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received <i>Reject</i> Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .

Table 8.147 Steps for UFP USB4 Mode Entry (Reject)

Step	DFP	UFP
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject</i> Message was successfully sent.
<i>Port Partners</i> do not enter <i>[USB4]</i> operation.		

This is an example of an Enter USB operation where the *DFP* requests *[USB4]* mode when this is not possible for the *UFP* at this time. [Figure 8.121, "UFP Entering USB4 Mode \(Wait\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter USB process.

Figure 8.121 UFP Entering USB4 Mode (Wait)



[Table 8.148, "Steps for UFP USB4 Mode Entry \(Wait\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.121, "UFP Entering USB4 Mode \(Wait\)"](#) above.

Table 8.148 Steps for UFP USB4 Mode Entry (Wait)

Step	DFP	UFP
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an <i>Enter_USB</i> Message to request entry to <i>[USB4]</i> mode.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the <i>Enter_USB</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Enter_USB</i> Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the <i>Enter_USB</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Enter_USB</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter_USB</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Wait</i> Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received <i>Wait</i> Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .

Table 8.148 Steps for UFP USB4 Mode Entry (Wait)

Step	DFP	UFP
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent.
<i>Port Partners</i> do not enter <i>[USB4]</i> operation.		

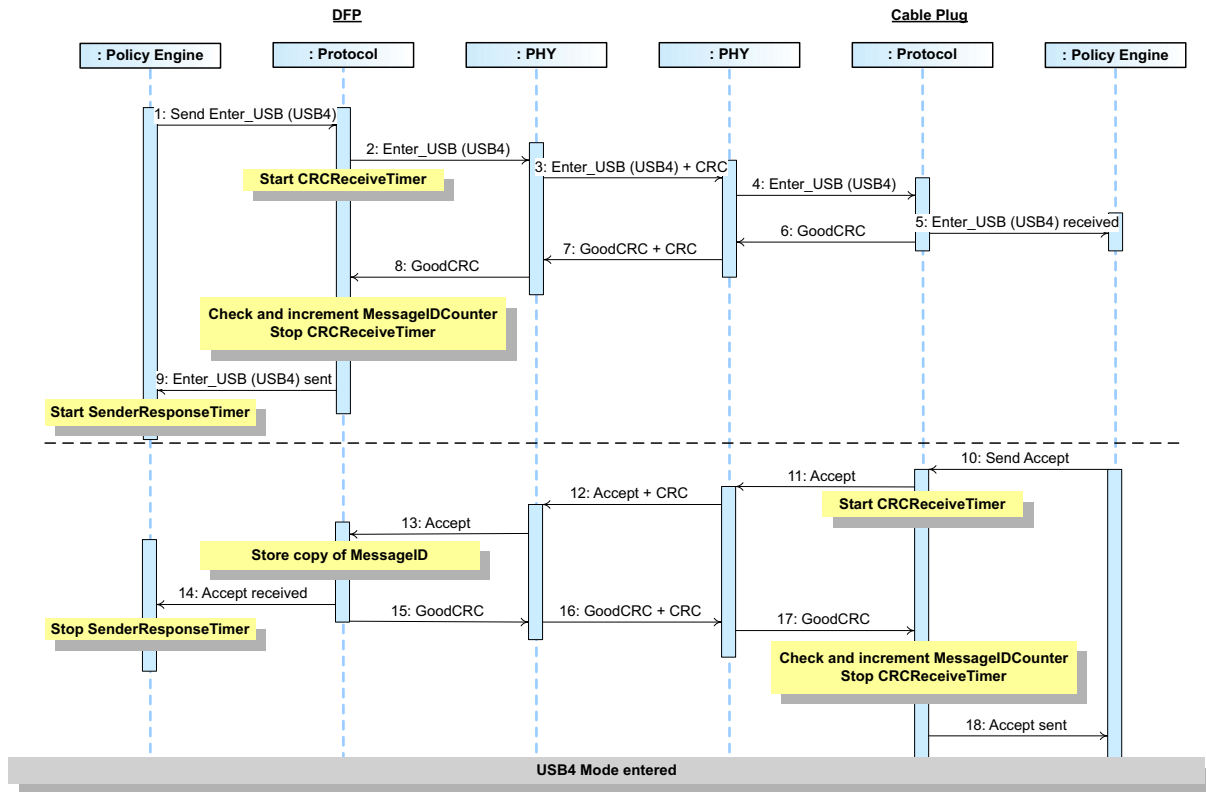
Cable Plug Entering USB4 Mode

8.3.2.16.2.1

Cable Plug Entering USB4 Mode (Accept)

This is an example of an Enter USB operation where the *DFP* requests **[USB4]** mode when this is a **Valid** mode of operation for the *Cable Plug*. [Figure 8.122, "Cable Plug Entering USB4 Mode \(Accept\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter USB process.

Figure 8.122 Cable Plug Entering USB4 Mode (Accept)



[Table 8.149, "Steps for Cable Plug USB4 Mode Entry \(Accept\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.122, "Cable Plug Entering USB4 Mode \(Accept\)"](#) above.

Table 8.149 Steps for Cable Plug USB4 Mode Entry (Accept)

Step	DFP	Cable Plug
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an Enter_USB Message to request entry to [USB4] mode.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Enter_USB Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Enter_USB Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Enter_USB Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Enter_USB Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Enter_USB Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Accept Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Accept Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .

Table 8.149 Steps for Cable Plug USB4 Mode Entry (Accept)

Step	DFP	Cable Plug
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Accept Message</i> was successfully sent.
<i>Cable Plug</i> enters <i>[USB4]</i> operation.		

Cable Plug Entering USB4 Mode (Reject)

[Table 8.150, "Steps for Cable Plug USB4 Mode Entry \(Reject\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.123, "Cable Plug Entering USB4 Mode \(Reject\)"](#) above.

Table 8.150 Steps for Cable Plug USB4 Mode Entry (Reject)

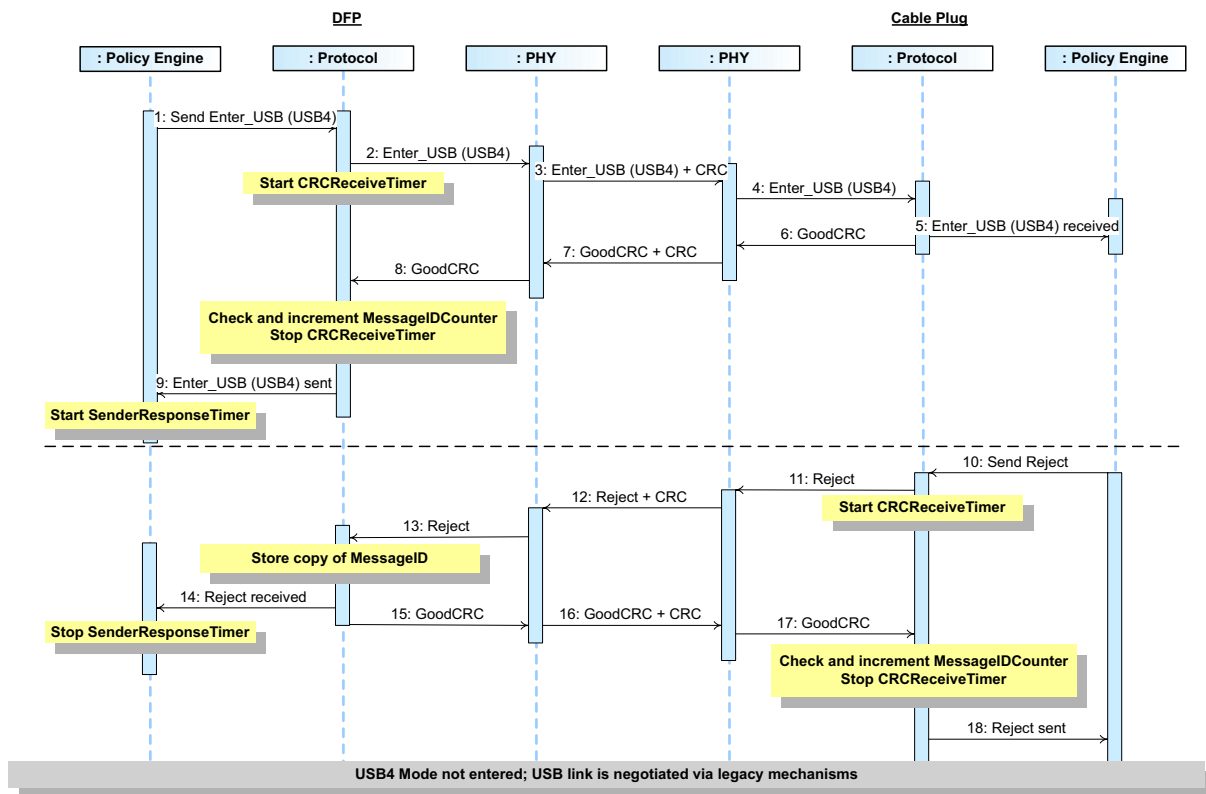
Step	DFP	Cable Plug
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an Enter_USB Message to request entry to [USB4] mode.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the Enter_USB Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Enter_USB Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the Enter_USB Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received Enter_USB Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Enter_USB Message was successfully sent. <i>Policy Engine</i> starts SenderResponseTimer .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Reject Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> stores the MessageID of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received Reject Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the GoodCRC Message.	<i>PHY Layer</i> receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the GoodCRC Message to the <i>Protocol Layer</i> .

Table 8.150 Steps for Cable Plug USB4 Mode Entry (Reject)

Step	DFP	Cable Plug
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Reject</i> Message was successfully sent.
<i>Cable Plug</i> does not enter <i>[USB4]</i> operation.		

This is an example of an Enter USB operation where the *DFP* requests *[USB4]* mode when this is not possible for the *Cable Plug* at this time. [Figure 8.124, "Cable Plug Entering USB4 Mode \(Wait\)"](#) shows the *Messages* as they flow across the bus and within the devices to accomplish the Enter USB process.

Figure 8.124 Cable Plug Entering USB4 Mode (Wait)



[Table 8.151, "Steps for Cable Plug USB4 Mode Entry \(Wait\)"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.124, "Cable Plug Entering USB4 Mode \(Wait\)"](#) above.

Table 8.151 Steps for Cable Plug USB4 Mode Entry (Wait)

Step	DFP	Cable Plug
1	The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to generate an <i>Enter_USB</i> Message to request entry to <i>[USB4]</i> mode.	
2	<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends CRC and sends the <i>Enter_USB</i> Message. Starts <i>CRCReceiveTimer</i> .	<i>PHY Layer</i> receives the <i>Enter_USB</i> Message and compares the CRC it calculated with the one sent to verify the Message.
4		<i>PHY Layer</i> removes the CRC and forwards the <i>Enter_USB</i> Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the <i>MessageID</i> in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the received <i>Enter_USB</i> Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the <i>GoodCRC</i> and checks the CRC to verify the Message.	<i>PHY Layer</i> appends CRC and sends the <i>GoodCRC</i> Message.
8	<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Enter_USB</i> Message was successfully sent. <i>Policy Engine</i> starts <i>SenderResponseTimer</i> .	
10		<i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an <i>Wait</i> Message.
11		<i>Protocol Layer</i> creates the Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Message and compares the CRC it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a CRC and sends the Message. Starts <i>CRCReceiveTimer</i> .
13	<i>Protocol Layer</i> stores the <i>MessageID</i> of the incoming Message.	
14	The <i>Protocol Layer</i> forwards the received <i>Wait</i> Message information to the <i>Policy Engine</i> that consumes it.	
15	<i>Protocol Layer</i> generates a <i>GoodCRC</i> Message and passes it <i>PHY Layer</i> .	
16	<i>PHY Layer</i> appends a CRC and sends the <i>GoodCRC</i> Message.	<i>PHY Layer</i> receives <i>GoodCRC</i> Message and compares the CRC it calculated with the one sent to verify the Message.
17		<i>PHY Layer</i> removes the CRC and forwards the <i>GoodCRC</i> Message to the <i>Protocol Layer</i> .

Table 8.151 Steps for Cable Plug USB4 Mode Entry (Wait)

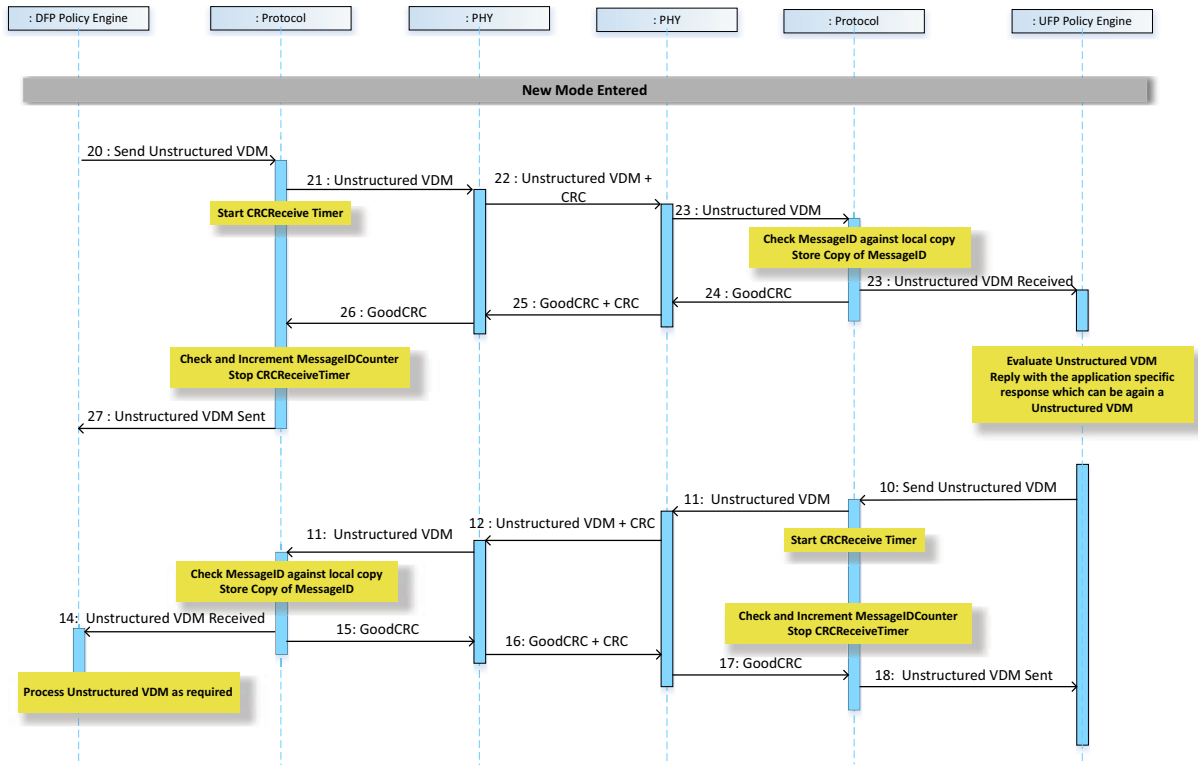
Step	DFP	Cable Plug
18		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Wait Message</i> was successfully sent.
<i>Cable Plug</i> does not enter <i>[USB4]</i> operation.		

8.3.2.17 Unstructured Vendor Defined Messages

8.3.2.17.1 Unstructured VDM

Figure 8.125, "Unstructured VDM Message Sequence" shows an example sequence of an *Unstructured VDM* Transaction between a *DFP* and *UFP*. The figure below shows the *Messages* as they flow across the bus after *UFP* Enters into *Modal Operation*.

Figure 8.125 Unstructured VDM Message Sequence



[Table 8.152, "Steps for Unstructured VDM Message Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.125, "Unstructured VDM Message Sequence"](#) above.

Table 8.152 Steps for Unstructured VDM Message Sequence

Step	DFP	UFP
1	The DFP has an <i>Explicit Contract</i> and has entered an <i>Active Mode</i> with the UFP. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send an Unstructured Vendor_Defined Message.	The UFP has an <i>Explicit Contract</i> and has entered an <i>Active Mode</i> with the UFP
2	<i>Protocol Layer</i> creates the Unstructured Vendor_Defined Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Unstructured Vendor_Defined Message. Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Unstructured Vendor_Defined Message and checks the <i>CRC</i> to verify the Message.
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Unstructured Vendor_Defined Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the Unstructured Vendor_Defined Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the Message.	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message.
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Unstructured Vendor_Defined Message was successfully sent.	
10		In this example the Vendor protocol requires a response. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form an Unstructured Vendor_Defined Message.
11		<i>Protocol Layer</i> creates the Unstructured Vendor_Defined Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Unstructured Vendor_Defined Message and compares the <i>CRC</i> it calculated with the one sent to verify the Message.	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Unstructured Vendor_Defined Message. Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming Message is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the Unstructured Vendor_Defined Message information to the <i>Policy Engine</i> that consumes it.	
14	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	

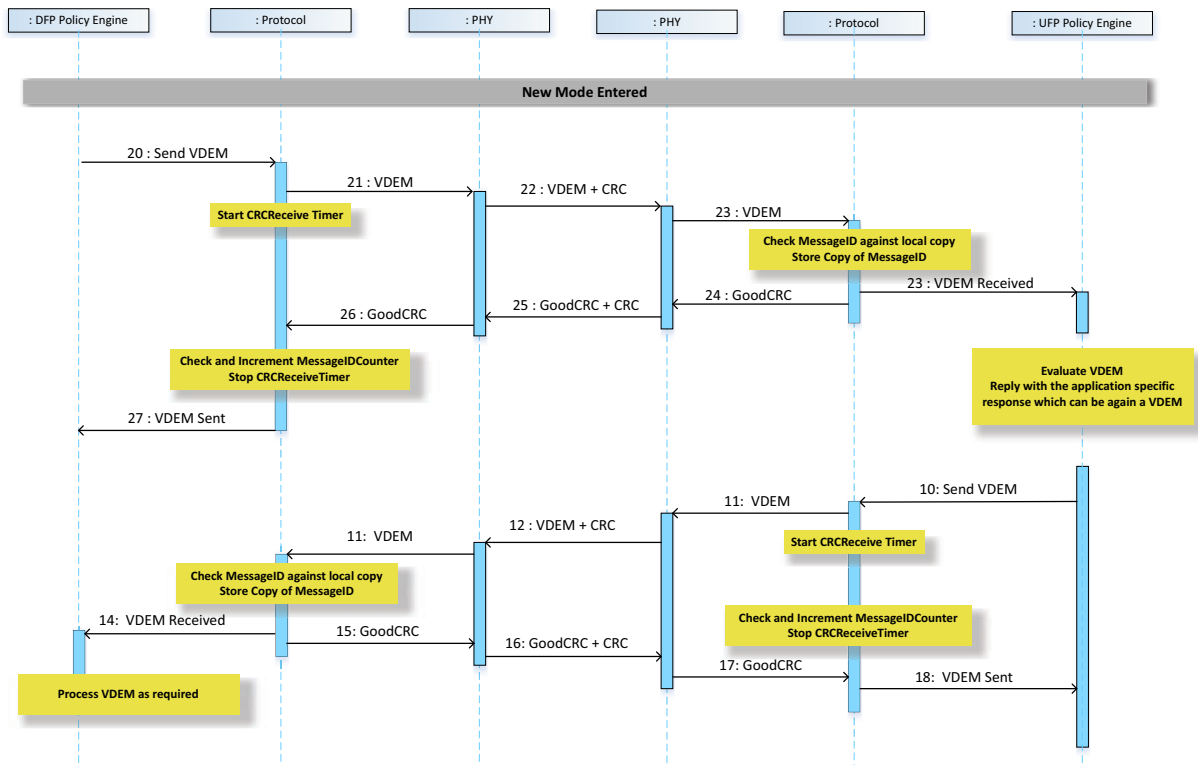
Table 8.152 Steps for Unstructured VDM Message Sequence

Step	DFP	UFP
15	<i>PHY Layer appends a CRC and sends the GoodCRC Message.</i>	<i>PHY Layer receives GoodCRC Message and compares the CRC it calculated with the one sent to verify the Message.</i>
16		<i>PHY Layer removes the CRC and forwards the GoodCRC Message to the Protocol Layer.</i>
17		<i>Protocol Layer verifies and increments the MessageIDCounter and stops CRCReceiveTimer. Protocol Layer informs the Policy Engine that the Unstructured Vendor_Defined Message was successfully sent.</i>

8.3.2.17.2 VDEM

Figure 8.126, "VDEM Message Sequence" shows an example sequence of an *VDEM* transaction between a *DFP* and *UFP*. The figure below shows the *Messages* as they flow across the bus after *UFP* Enters into *Modal Operation*.

Figure 8.126 VDEM Message Sequence



[Table 8.153, "Steps for VDEM Message Sequence"](#) below provides a detailed explanation of what happens at each labeled step in [Figure 8.126, "VDEM Message Sequence"](#) above.

Table 8.153 Steps for VDEM Message Sequence

Step	DFP	UFP
1	The DFP has an <i>Explicit Contract</i> and has entered an <i>Active Mode</i> with the UFP. The <i>Policy Engine</i> directs the <i>Protocol Layer</i> to send a Vendor_Defined_Extended Message .	The UFP has an <i>Explicit Contract</i> and has entered an <i>Active Mode</i> with the UFP
2	<i>Protocol Layer</i> creates the Vendor_Defined_Extended Message and passes to <i>PHY Layer</i> .	
3	<i>PHY Layer</i> appends <i>CRC</i> and sends the Vendor_Defined_Extended Message . Starts CRCReceiveTimer .	<i>PHY Layer</i> receives the Vendor_Defined_Extended Message and checks the <i>CRC</i> to verify the <i>Message</i> .
4		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the Vendor_Defined_Extended Message to the <i>Protocol Layer</i> .
5		<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the Vendor_Defined_Extended Message information to the <i>Policy Engine</i> that consumes it.
6		<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .
7	<i>PHY Layer</i> receives the GoodCRC Message and checks the <i>CRC</i> to verify the <i>Message</i> .	<i>PHY Layer</i> appends <i>CRC</i> and sends the GoodCRC Message .
8	<i>PHY Layer</i> removes the <i>CRC</i> and forwards the GoodCRC Message to the <i>Protocol Layer</i> .	
9	<i>Protocol Layer</i> verifies and increments the MessageIDCounter and stops CRCReceiveTimer . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the Vendor_Defined_Extended Message was successfully sent.	
10		In this example the Vendor protocol requires a response. The <i>Policy Engine</i> tells the <i>Protocol Layer</i> to form a Vendor_Defined_Extended Message .
11		<i>Protocol Layer</i> creates the Vendor_Defined_Extended Message and passes to <i>PHY Layer</i> .
12	<i>PHY Layer</i> receives the Vendor_Defined_Extended Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .	<i>PHY Layer</i> appends a <i>CRC</i> and sends the Vendor_Defined_Extended Message . Starts CRCReceiveTimer .
13	<i>Protocol Layer</i> checks the MessageID in the incoming <i>Message</i> is different from the previously stored value and then stores a copy of the new value. The <i>Protocol Layer</i> forwards the Vendor_Defined_Extended Message information to the <i>Policy Engine</i> that consumes it.	
14	<i>Protocol Layer</i> generates a GoodCRC Message and passes it <i>PHY Layer</i> .	
15	<i>PHY Layer</i> appends a <i>CRC</i> and sends the GoodCRC Message .	<i>PHY Layer</i> receives GoodCRC Message and compares the <i>CRC</i> it calculated with the one sent to verify the <i>Message</i> .

Table 8.153 Steps for VDEM Message Sequence

Step	DFP	UFP
16		<i>PHY Layer</i> removes the <i>CRC</i> and forwards the <i>GoodCRC Message</i> to the <i>Protocol Layer</i> .
17		<i>Protocol Layer</i> verifies and increments the <i>MessageIDCounter</i> and stops <i>CRCReceiveTimer</i> . <i>Protocol Layer</i> informs the <i>Policy Engine</i> that the <i>Vendor_Defined_Extended Message</i> was successfully sent.

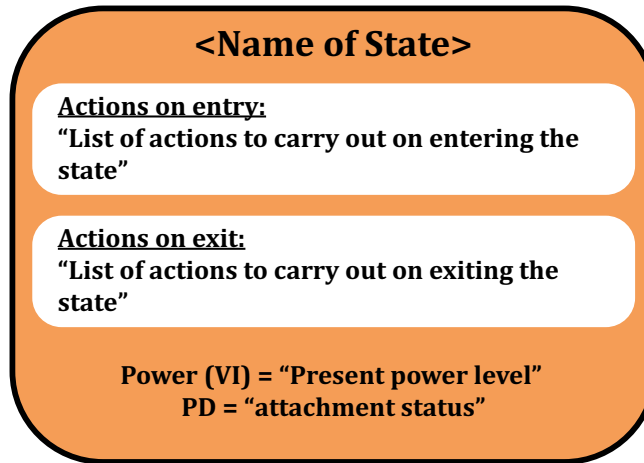
8.3.3 State Diagrams

8.3.3.1 Introduction to state diagrams used in Chapter 8

The state diagrams defined in [Section 8.3.3, "State Diagrams"](#) are **Normative** and **Shall** define the operation of the Power Delivery Policy Engine.

Note: These state diagrams are not intended to replace a well written and robust design.

Figure 8.127 Outline of States



[Figure 8.127, "Outline of States"](#) shows an outline of the states defined in the following sections. At the top there is the name of the state. This is followed by "Actions on entry" a list of actions carried out on entering the state. If there are also "Actions on exit" a list of actions carried out on exiting the state, then these are listed as well; otherwise, this box is omitted from the state. At the bottom the status of PD is listed:

- "Power" which indicates the present output power for a *Source Port* or input power for a *Sink Port*.
- "PD" which indicates the present *Attachment* status either "Attached", "Detached", or "unknown".

Transitions from one state to another are indicated by arrows with the conditions listed on the arrow. Where there are multiple conditions, these are connected using either a logical OR "|" or a logical AND "&".

In some cases, there are transitions which can occur from any state to a particular state. These are indicated by an arrow which is unconnected to a state at one end, but with the other end (the point) connected to the final state.

In some state diagrams it is necessary to enter or exit from states in other diagrams (e.g., *Source Port* or *Sink Port* state diagrams). [Figure 8.128, "References to states"](#) indicates how such references are made. The reference is indicated with a hatched box. The box contains the name of the state and whether the state is a *DFP* or *UFP*. It has also been necessary to indicate conditional entry to either *Source Port* or *Sink Port* state diagrams. This is achieved by the use of a bulleted list indicating the preconditions (see example in [Figure 8.129, "Example of state reference with conditions"](#)). It is also possible that the entry and return states are the same. [Figure 8.130, "Example of state reference with the same entry and exit"](#) indicates a state reference where each referenced state corresponds to either the entry state or the exit state.

Figure 8.128 References to states

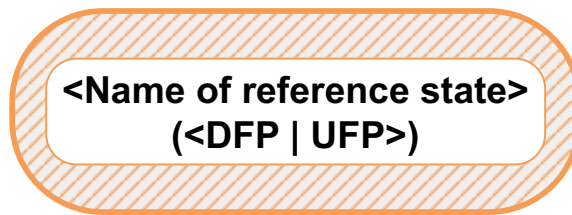


Figure 8.129 Example of state reference with conditions

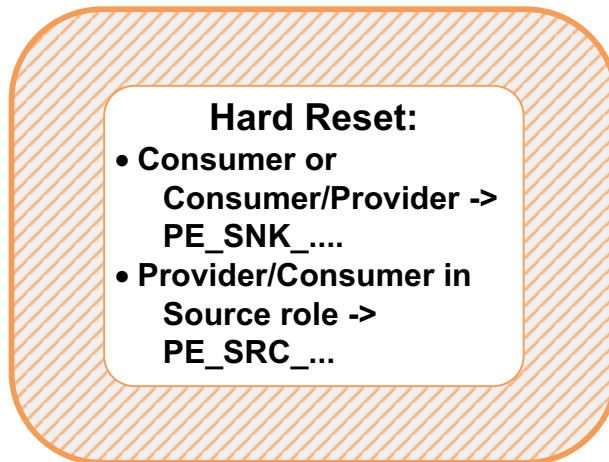
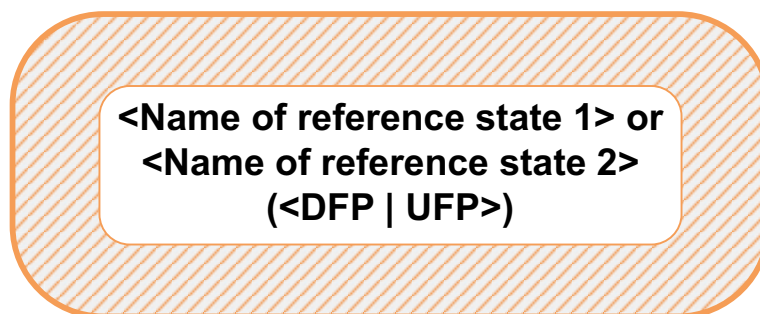


Figure 8.130 Example of state reference with the same entry and exit



Timers are included in many of the states. Timers are initialized (set to their starting condition) and run (timer is counting) in the particular state it is referenced. As soon as the state is exited then the timer is no longer active. Where the timers continue to run outside of the state (such as the *NoResponseTimer*), this is called out in the text. Timeouts of the timers are listed as conditions on state transitions.

The *SenderResponseTimer* is a special case, as it **May** be stopped and started from outside the states in which it is used. To allow this to be done without over-complicating the state diagrams, the *SenderResponseTimer* is described with its own state diagram ([Figure 8.131, "SenderResponseTimer Policy Engine State Diagram"](#)). The control of this Timer is shared between the *Policy Engine* and the *Chunking Layer*.

Conditions listed on state transitions will come from one of three sources and, when there is a conflict, **Should** be serviced in the following order:

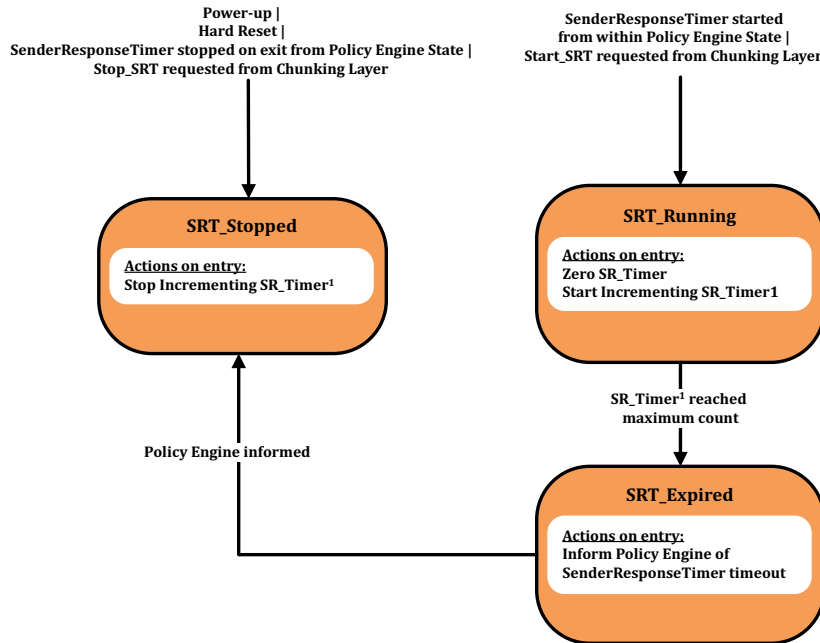
- 1) *Message* and related indications passed up to the *Policy Engine* from the *Protocol Layer* (*Message* sent; *Message* received etc.).
- 2) Events triggered within the *Policy Engine* e.g., timer timeouts.
- 3) Information and requests coming from the *Device Policy Manager* relating either to *Local Policy*, or to other modules which the *Device Policy Manager* controls such as power supply and USB-C[®] Port Control.

Note: The following state diagrams are not intended to cover all possible corner cases that could be encountered. For example, where an outgoing *Message* is **Discarded**, due to an incoming *Message* by the *Protocol Layer* (see [Section 6.12.2.3, "Protocol Layer Message Reception"](#)) it will be necessary for the higher layers of the system to handle a retry of the *AMS* that was being initiated, after first handling the incoming *Message*.

8.3.3.1.1 SenderResponseTimer State Diagram

Figure 8.131, "*SenderResponseTimer Policy Engine State Diagram*" below shows the state diagram for the *Policy Engine* in a *Source Port* or a *Sink Port*. The following sections describe operation in each of the states.

Figure 8.131 SenderResponseTimer Policy Engine State Diagram



¹⁾ The SR_Timer is regarded as the mechanism within the *SenderResponseTimer* state diagram that implements the *SenderResponseTimer*.

8.3.3.1.1.1 SRT_Stopped State

The **SRT_Stopped** State **Shall** be the starting state for the *SenderResponseTimer* either on power up or after a *Hard Reset*. On entry to this state the *Policy Engine* **Shall** stop incrementing the SR_Timer.

The *Policy Engine* **Shall** transition to the **SRT_Running** State:

- When the *SenderResponseTimer* is started from within a *Policy Engine* state, or
- When a Start_SRT is requested from the *Chunking Layer*.

8.3.3.1.1.2 SRT_Running State

On entry to the **SRT_Running** State the *SenderResponseTimer* state machine **Shall**:

- Set the SR_Timer to zero
- Start running SR_Timer.

The *SenderResponseTimer* state machine **Shall** transition to the **SRT_Expired** State:

- When the SR_Timer reaches its maximum count

The *SenderResponseTimer* state machine **Shall** transition to the **SRT_Stopped** State:

- When the *SenderResponseTimer* is stopped by exiting a *Policy Engine* state, or
- When a Stop_SRT is requested from the *Chunking Layer*

On entry to the **SRT_Running** State the *SenderResponseTimer* state machine **Shall** Inform *Policy Engine* of *SenderResponseTimer* timeout

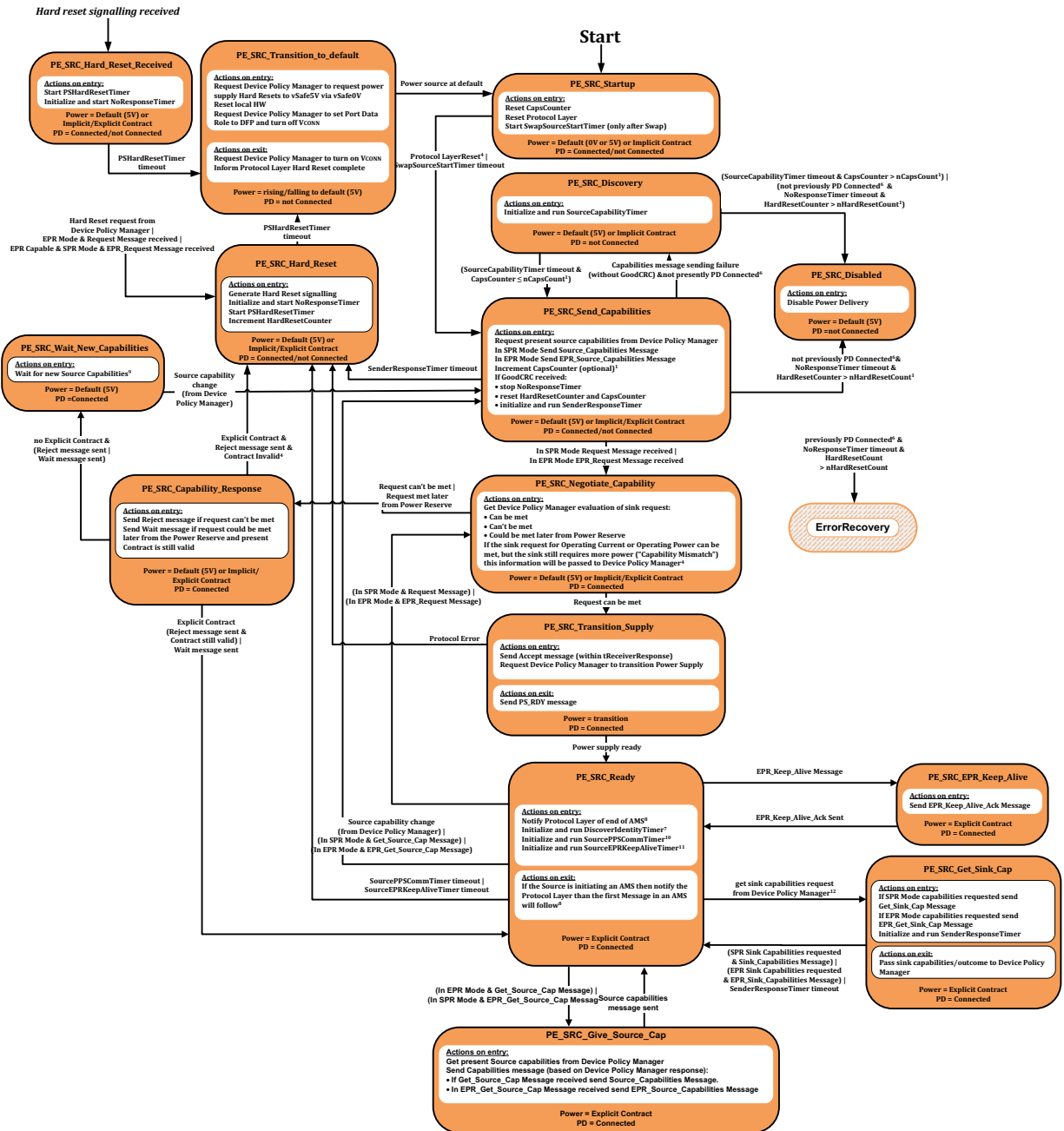
The *Policy Engine* **Shall** then transition to the **SRT_Stopped** state:

- When the *Policy Engine* has been informed.

8.3.3.2 Policy Engine Source Port State Diagram

Figure 8.132, "Source Port State Diagram" below shows the state diagram for the Policy Engine in a Source Port. The following sections describe operation in each of the states.

Figure 8.132 Source Port State Diagram



- 1) Implementation of the **CapsCounter** is **Optional**. In the case where this is not implemented the **Source** **Shall** continue to send **Source_Capabilities** Messages each time the **SourceCapabilityTimer** times out.
- 2) Since the **Sink** is required to make a **Valid** request from the offered capabilities the expected transition is via "Request can be met" unless the **Source Capabilities** have changed since the last offer.
- 3) "Contract Invalid" means that the previously **Negotiated** voltage and Current values are no longer included in the **Source's** new Capabilities. If the **Sink** fails to make a **Valid** Request in this case, then Power Delivery operation is no longer possible and Power Delivery mode is exited with a **Hard Reset**.

- 4) After a Power Swap the *New Source* is required to wait an additional ***tSwapSourceStart*** before sending a ***Source_Capabilities*** Message. This delay is not required when first starting up a system.
- 5) *PD Connected* is defined as a situation when the *Port Partners* are actively communicating. The *Port Partners* remain *PD Connected* after a Swap until there is a transition to Disabled or the connector is able to identify a *Detach*.
- 6) *Port Partners* are no longer *PD Connected* after a *Hard Reset*, but consideration needs to be given as to whether there has been a *PD Connection* while the Ports have been *Attached* to prevent unnecessary *USB Type-C Error Recovery*.
- 7) The ***DiscoverIdentityTimer*** is run when this is a *VCONN Source* and a *PD Connection* with a *Cable Plug* needs to be established i.e. no ***GoodCRC*** Message has yet been received in response to a ***Discover Identity*** Command.
- 8) See [Section 5.7, "Collision Avoidance"](#), [Section 6.6.16, "Collision Avoidance Timers"](#) and [Section 6.10, "Collision Avoidance"](#).
- 9) In the ***PE_SRC_Wait_New_Capabilities*** State the *Device Policy Manager* **Should** either decide to send no further *Source Capabilities* or **Should** send a different set of *Source Capabilities*. Continuing to send the same set of *Source Capabilities* could result in a live lock situation.
- 10) The ***SourcePPSCommTimer*** is only initialized and run when the present *Explicit Contract* is for an *SPR PPS APDO*. Sources that do not support *SPR PPS* do not need to implement the ***SourcePPSCommTimer***.
- 11) The ***SourceEPRKeepAliveTimer*** is only initialized and run when the *Source* is in *EPR Mode*; Sources that do not support *EPR Mode* do not need to implement the ***SourceEPRKeepAliveTimer***.
- 12) Either *SPR* or *EPR Sink Capabilities* **May** be requested, regardless of whether or not the *Source* is currently operating in *SPR* or *EPR Mode*.

8.3.3.2.1 PE_SRC_Startup State

PE_SRC_Startup **Shall** be the starting state for a *Source Policy Engine* either on power up or after a *Hard Reset*. On entry to this state the *Policy Engine* **Shall** reset the ***CapsCounter*** and reset the *Protocol Layer*.

Note: Resetting the *Protocol Layer* will also reset the ***MessageIDCounter*** and stored ***MessageID*** (see [Section 6.12.2.3, "Protocol Layer Message Reception"](#)).

The *Policy Engine* **Shall** transition to the ***PE_SRC_Send_Capabilities*** state:

- When the *Protocol Layer* reset has completed if the ***PE_SRC_Startup*** state was entered due to the system first starting up.
- When the ***SwapSourceStartTimer*** times out if the ***PE_SRC_Startup*** state was entered as the result of a *Power Role Swap*.

Note: Sources **Shall** remain in the ***PE_SRC_Startup*** state, without sending any ***Source_Capabilities*** Messages until a plug is *Attached*.

8.3.3.2.2 PE_SRC_Discovery State

On entry to the ***PE_SRC_Discovery*** state the *Policy Engine* **Shall** initialize and run the ***SourceCapabilityTimer*** in order to trigger sending a ***Source_Capabilities*** Message.

The *Policy Engine* **Shall** transition to the ***PE_SRC_Send_Capabilities*** state when:

- The ***SourceCapabilityTimer*** times out and ***CapsCounter* ≤ *nCapsCount***.

The *Policy Engine* **May Optionally** go to the ***PE_SRC_Disabled*** state when:

- The *Port Partners* are not presently *PD Connected*
- And the ***SourceCapabilityTimer*** times out
- And ***CapsCounter* > *nCapsCount***.

The *Policy Engine* **Shall** go to the ***PE_SRC_Disabled*** state when:

- The *Port Partners* have not been *PD Connected* (the *Source Port* remains *Attached* to a Port it has not had a *PD Connection* with during this *Attachment*)

- And the **NoResponseTimer** times out
- And the **HardResetCounter** > **nHardResetCount**.

Note: In the **PE_SRC_Disabled** state the *Attached* device is assumed to be unresponsive. The *Policy Engine* operates as if the device is *Detached* until such time as a *Detach/Re-attach* is detected.

8.3.3.2.3 PE_SRC_Send_Capabilities State

Note: This state can be entered from the **PE_SRC_Soft_Reset** state.

On entry to the **PE_SRC_Send_Capabilities** state the *Policy Engine* **Shall** request the present Port capabilities from the *Device Policy Manager*. The *Policy Engine* **Shall** then request the *Protocol Layer* to send a capabilities *Message* containing these capabilities. The *Policy Engine* **Shall** request:

- A **Source_Capabilities** *Message* if the *Source* is in *SPR Mode* or
- An **EPR_Source_Capabilities** *Message* if the *Source* is in *EPR Mode*.

The *Policy Engine* **Shall** then increment the **CapsCounter** (if implemented).

If a **GoodCRC** *Message* is received, then the *Policy Engine* **Shall**:

- Stop the **NoResponseTimer**.
- Reset the **HardResetCounter** and **CapsCounter** to zero.

Note: The **HardResetCounter** **Shall** only be set to zero in this state and at power up; its value **Shall** be maintained during a *Hard Reset*.

- Initialize and run the **SenderResponseTimer**.

Once a **Source_Capabilities** *Message* has been received and acknowledged by a **GoodCRC** *Message*, the *Sink* is required to then send a **Request** *Message* within **tSenderResponse**.

The *Policy Engine* **Shall** transition to the **PE_SRC_Negotiate_Capability** state when:

- A **Request** *Message* is received from the *Sink* and the *Source* is operating in *SPR Mode* or
- An **EPR_Request** *Message* is received from the *Sink* and the *Source* is operating in *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Discovery** state when:

- The *Protocol Layer* indicates that the *Message* has not been sent and we are presently not *Connected*. This is part of the Capabilities sending process whereby successful *Message* sending indicates connection to a PD *Sink Port*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- The **SenderResponseTimer** times out. In this case a transition back to *USB Default Operation* is required.

When:

- The *Port Partners* have not been PD *Connected* (the *Source Port* remains *Attached* to a Port it has not had a PD Connection with during this *Attachment*)
- And the **NoResponseTimer** times out
- And the **HardResetCounter** > **nHardResetCount**.

The *Policy Engine* **Shall** do one of the following:

- Transition to the **PE_SRC_Discovery** state.
- Transition to the **PE_SRC_Disabled** state.

Note: That in either case the *Attached* device is assumed to be unresponsive. The *Policy Engine* **Should** operate as if the device is *Detached* until such time as a *Detach/Re-attach* is detected.

The *Policy Engine* **Shall** go to the **ErrorRecovery** state when:

- The *Port Partners* have previously been PD *Connected* (the *Source Port* remains *Attached* to a Port it has had a PD Connection with during this *Attachment*)
- And the **NoResponseTimer** times out.
- And the **HardResetCounter** > **nHardResetCount**.

8.3.3.2.4 PE_SRC_Negotiate_Capability State

On entry to the **PE_SRC_Negotiate_Capability** state the *Policy Engine* **Shall** ask the *Device Policy Manager* to evaluate the Request from the *Attached Sink*. The response from the *Device Policy Manager* **Shall** be one of the following:

- The Request can be met.
- The Request cannot be met
- The Request could be met later from the Power Reserve.

The *Policy Engine* **Shall** transition to the **PE_SRC_Transition_Supply** state when:

- The Request can be met.

The *Policy Engine* **Shall** transition to the **PE_SRC_Capability_Response** state when:

- The Request cannot be met.
- Or the Request can be met later from the Power Reserve.

8.3.3.2.5 PE_SRC_Transition_Supply State

The *Policy Engine* **Shall** be in the **PE_SRC_Transition_Supply** state while the power supply is transitioning from one power to another.

On entry to the **PE_SRC_Transition_Supply** state, the *Policy Engine* **Shall** request the *Protocol Layer* to send an **Accept** Message and inform the *Device Policy Manager* that it **Shall** transition the power supply to the Requested power level.

Note: If the power supply is currently operating at the requested power no change will be necessary.

On exit from the **PE_SRC_Transition_Supply** state the *Policy Engine* **Shall** request the *Protocol Layer* to send a **PS_RDY** Message.

The *Policy Engine* **Shall** transition to the **PE_SRC_Ready** state when:

- The *Device Policy Manager* informs the *Policy Engine* that the power supply is ready.

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- A *Protocol Error* occurs.

8.3.3.2.6 PE_SRC_Ready State

In the **PE_SRC_Ready** state the PD *Source* **Shall** be operating at a stable power with no ongoing *Negotiation*. It **Shall** respond to requests from the *Sink*, events from the *Device Policy Manager*.

On entry to the **PE_SRC_Ready** state the *Source* **Shall** notify the *Protocol Layer* of the end of the *Atomic Message Sequence* (AMS). If the transition into **PE_SRC_Ready** is the result of *Protocol Error* that has not caused a *Soft Reset* (see [Section 8.3.3.4.1, "SOP Source Port Soft Reset and Protocol Error State Diagram"](#)) then the notification to the *Protocol Layer* of the end of the AMS **Shall Not** be sent since there is a *Message* to be processed.

On entry to the **PE_SRC_Ready** state if this is a *VCONN Source* which needs to establish communication with a *Cable Plug*, the *Policy Engine* **Shall**:

- Initialize and run the **DiscoverIdentityTimer** (no **GoodCRC** Message response yet received to **Discover Identity** Message).

On entry to the **PE_SRC_Ready** state if the current *Explicit Contract* is for an *SPR PPS APDO*, then the *Policy Engine* **Shall** do the following:

- Initialize and run the **SourcePPSCommTimer**.

On entry to the **PE_SRC_Ready** state if the current *Explicit Contract* is for *EPR Mode*, then the *Policy Engine* **Shall** do the following:

- Initialize and run the **SourceEPRKeepAliveTimer**.

On exit from the **PE_SRC_Ready**, if the *Source* is initiating an *AMS*, then the *Policy Engine* **Shall** notify the *Protocol Layer* that the first *Message* in an *AMS* will follow.

The *Policy Engine* **Shall** transition to the **PE_SRC_Send_Capabilities** state when:

- The *Device Policy Manager* indicates that *Source Capabilities* have changed or
- A **Get_Source_Cap** Message is received, and the *Source* is in *SPR Mode* or
- An **EPR_Get_Source_Cap** Message is received, and the *Source* is in *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Negotiate_Capability** state when:

- A **Request** Message is received, and the *Source* is in *SPR Mode* or
- An **EPR_Request** Message is received, and the *Source* is in *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Get_Sink_Cap** state when:

- The *Device Policy Manager* asks for the *Sink Capabilities*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- The *Source* is operating as an *SPR PPS* and the **SourcePPSCommTimer** Timer times-out or
- The *Source* is in *EPR Mode* and the **SourceEPRKeepAliveTimer** Timer times-out.

The *Policy Engine* **Shall** transition to the **PE_SRC_EPR_Keep_Alive** state when:

- An **EPR_KeepAlive** Message is received.

The *Policy Engine* **Shall** transition to the **PE_SRC_Give_Source_Cap** State when:

- In *EPR Mode* and a **Get_Source_Cap** Message is received or
- In *SPR Mode* and an **EPR_Get_Source_Cap** Message is received.

8.3.3.2.7 PE_SRC_Disabled State

In the **PE_SRC_Disabled** state the *PD Source* supplies default power and is unresponsive to *USB Power Delivery* messaging, but not to **Hard Reset Signaling**.

8.3.3.2.8 PE_SRC_Capability_Response State

The *Policy Engine* **Shall** enter the **PE_SRC_Capability_Response** state if there is a *Request* received from the *Sink* that cannot be met based on the present capabilities. When the present *Explicit Contract* is not within the present capabilities it is regarded as **Invalid** and a *Hard Reset* will be triggered.

On entry to the **PE_SRC_Hard_Reset** state the *Policy Engine* **Shall** request the *Protocol Layer* to send one of the following:

- **Reject** Message - if the request cannot be met or the present *Explicit Contract* is **Invalid**.
- **Wait** Message - if the request could be met later from the Power Reserve. A **Wait** Message **Shall Not** be sent if the present *Explicit Contract* is **Invalid**.

The *Policy Engine* **Shall** transition to the **PE_SRC_Ready** state when:

- There is an *Explicit Contract* and
- A **Reject** Message has been sent and the present *Explicit Contract* is still **Valid** or
- A **Wait** Message has been sent.

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- There is an *Explicit Contract* and
- The **Reject** Message has been sent and the present *Explicit Contract* is **Invalid** (i.e., the *Sink* had to request a new value so instead we will return to *USB Default Operation*).

The *Policy Engine* **Shall** transition to the **PE_SRC_Wait_New_Capabilities** state when:

- There is no *Explicit Contract* and
- A **Reject** Message has been sent or
- A **Wait** Message has been sent.

8.3.3.2.9 PE_SRC_Hard_Reset State

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state from any state when:

- *Hard Reset* request from *Device Policy Manager* or
- In *EPR Mode* and a **Request** Message is received or
- *EPR Capable* and in *SPR Mode* and an **EPR_Request** Message is received.

On entry to the **PE_SRC_Hard_Reset** state the *Policy Engine* **Shall**:

- request the generation of **Hard Reset** Signaling by the PHY Layer
- initialize and run the **NoResponseTimer**.

Note: The **NoResponseTimer** **Shall** continue to run in every state until it is stopped or times out.

- initialize and run the **PSHardResetTimer** and increment the **HardResetCounter**.

The *Policy Engine* **Shall** transition to the **PE_SRC_Transition_to_default** state when:

- The **PSHardResetTimer** times out.

8.3.3.2.10 PE_SRC_Hard_Reset_Received State

The *Policy Engine* **Shall** transition from any state to the **PE_SRC_Hard_Reset_Received** state when:

- **Hard Reset** Signaling is detected.

On entry to the **PE_SRC_Hard_Reset_Received** state the *Policy Engine* **Shall**:

- initialize and run the **PSHardResetTimer**
- initialize and run the **NoResponseTimer**.

Note: The **NoResponseTimer** **Shall** continue to run in every state until it is stopped or times out.

The *Policy Engine* **Shall** transition to the **PE_SRC_Transition_to_default** state when:

- The *PSHardResetTimer* times out.

8.3.3.2.11 PE_SRC_Transition_to_default State

On entry to the *PE_SRC_Transition_to_default* state the *Policy Engine* **Shall**:

- indicate to the *Device Policy Manager* that the power supply **Shall** Hard Reset (see [Section 7.1.5. "Response to Hard Resets"](#)).
- request a reset of the local hardware
- request the *Device Policy Manager* to set the *Port Data Role* to *DFP* and turn off *VCONN*.

On exit from the *PE_SRC_Transition_to_default* state the *Policy Engine* **Shall**:

- request the *Device Policy Manager* to turn on *VCONN*
- inform the *Protocol Layer* that the *Hard Reset* is complete.

The *Policy Engine* **Shall** transition to the *PE_SRC_Startup* state when:

- The *Device Policy Manager* indicates that the power supply has reached the default level.

8.3.3.2.12 PE_SRC_Get_Sink_Cap State

In this state the *Policy Engine*, due to a request from the *Device Policy Manager*, **Shall** request the capabilities from the *Attached Sink*.

On entry to the *PE_SRC_Get_Sink_Cap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send a *Get_Sink_Cap* Message in order to retrieve the *Sink Capabilities*. The *Policy Engine* **Shall** send:

- A *Get_Sink_Cap* Message when the *Device Policy Manager* requests *SPR capabilities* or
- An *EPR_Get_Sink_Cap* Message when the *Device Policy Manager* requests *EPR Capabilities*.

The *Policy Engine* **Shall** then start the *SenderResponseTimer*.

On exit from the *PE_SRC_Get_Sink_Cap* state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

The *Policy Engine* **Shall** transition to the *PE_SRC_Ready* state when:

- *SPR Sink Capabilities* were requested and a *Sink_Capabilities* Message is received or
- *EPR Sink Capabilities* were requested and an *EPR_Sink_Capabilities* Message is received or
- The *SenderResponseTimer* times out.

8.3.3.2.13 PE_SRC_Wait_New_Capabilities State

In this state the *Policy Engine* has been unable to *Negotiate* an *Explicit Contract* and is waiting for new *Capabilities* from the *Device Policy Manager*.

The *Policy Engine* **Shall** transition to the *PE_SRC_Send_Capabilities* state when:

- The *Device Policy Manager* indicates that *Source Capabilities* have changed.

8.3.3.2.14 PE_SRC_EPR_Keep_Alive State

On entry to the *PE_SRC_EPR_Keep_Alive* State the *Policy Engine* **Shall** send a *EPR_KeepAlive_Ack* Message.

The *Policy Engine* **Shall** transition to the *PE_SRC_Ready* state when:

- The *EPR_KeepAlive_Ack* Message has been sent.

- On entry to the **PE_SRC_Give_Source_Cap** State the *Policy Engine* **Shall** request the *Device Policy Manager* for the current system capabilities.

The *Policy Engine* **Shall** then request the *Protocol Layer* to send a *Source Capabilities Message* containing these capabilities.

The *Policy Engine* **Shall** send:

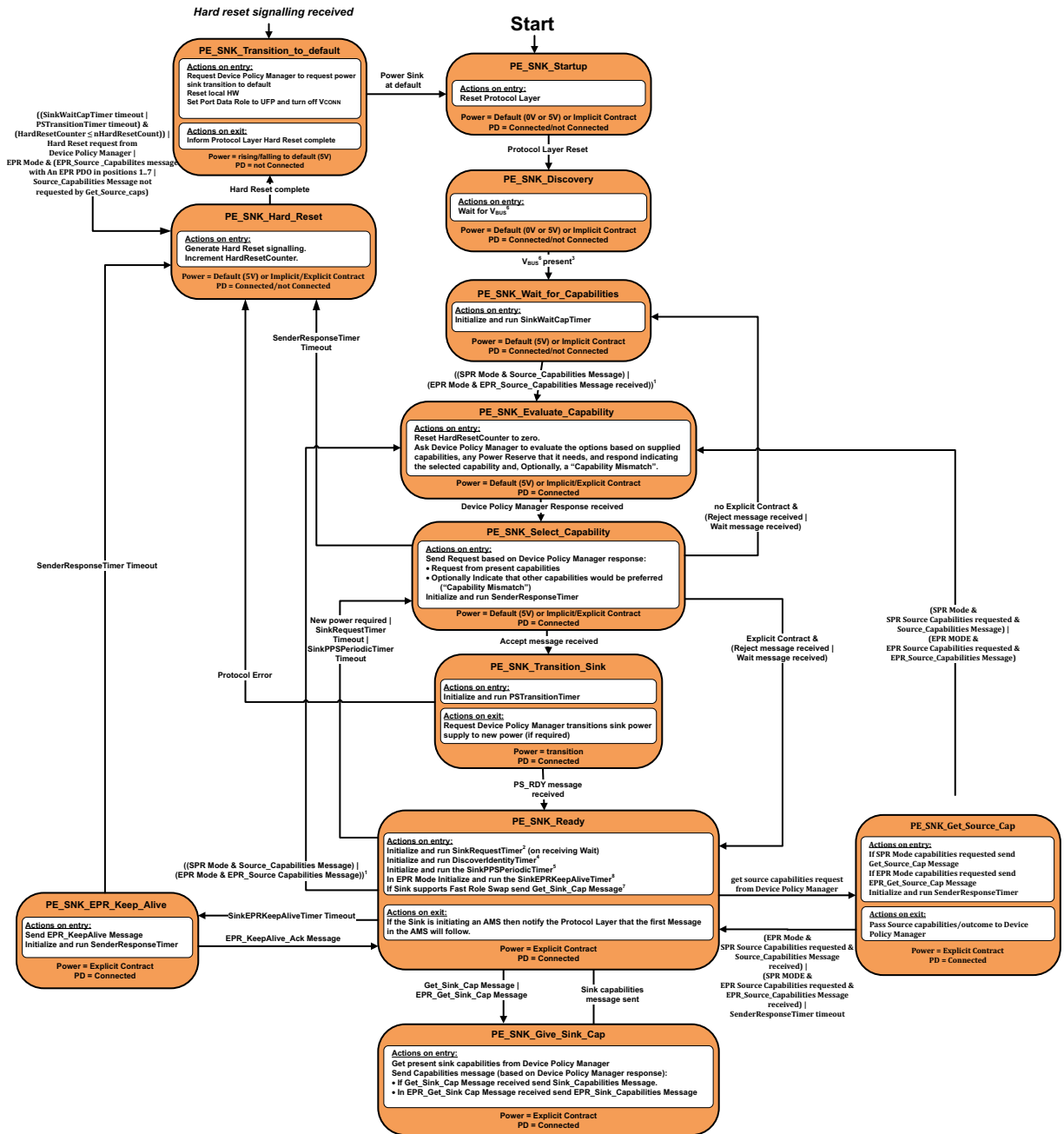
- A **Source_Capabilities** Message when a **Get_Source_Cap** Message is received or
- An **EPR_Source_Capabilities** Message when a **EPR_Get_Source_Cap** Message is received.

The *Policy Engine* **Shall** transition to the **PE_SNK_Ready** state when:

- The *Source Capabilities Message* has been successfully sent.

Figure 8.133, "Sink Port State Diagram" below shows the state diagram for the Policy Engine in a Sink Port. The following sections describe operation in each of the states.

Figure 8.133 Sink Port State Diagram



- 1) Source Capabilities Messages received in States other than **PE_SNK_Wait_for_Capabilities**, **PE_SNK_Ready** or **PE_SNK_Get_Source_Cap** constitute a Protocol Error.
- 2) The *SinkRequestTimer* Should Not be stopped if a *Ping (Deprecated)* Message is received in the **PE_SNK_Ready** state since it represents the maximum time between requests after a *Wait Message* which is not reset by a *Ping (Deprecated)* Message.
- 3) During a *Hard Reset* the Source voltage will transition to *vSafe0V* and then transition to *vSafe5V*. Sinks need to ensure that *V_{BUS}* present is not indicated until after the Source has completed the *Hard Reset* process by detecting both of these transitions.

- 4) The **DiscoverIdentityTimer** is run when this is a *VCONN Source* and a PD Connection with a *Cable Plug* needs to be established i.e. no **GoodCRC Message** has yet been received in response to a **Discover Identity Command**.
- 5) The **SinkPPSPeriodicTimer** is only initialized and run when the present *Explicit Contract* is for an *SPR PPS APDO*. *Sinks* that do not support PPS do not need to implement the **SinkPPSPeriodicTimer**.
- 6) A *Sink* that is a *VPD May* use *VCONN* as a proxy for *VBUS*.
- 7) To be sent once, and only required if *Fast Role Swap* is supported by the *Sink*.

8.3.3.3.1 PE_SNK_Startup State

PE_SNK_Startup Shall be the starting state for a *Sink Policy Engine* either on power up or after a *Hard Reset*. On entry to this state the *Policy Engine Shall* reset the *Protocol Layer*.

Note: Resetting the *Protocol Layer* will also reset the **MessageIDCounter** and stored **MessageID** (see [Section 6.12.2.3, "Protocol Layer Message Reception"](#)).

Once the reset process completes, the *Policy Engine Shall* transition to the **PE_SNK_Discovery** state.

8.3.3.3.2 PE_SNK_Discovery State

In the **PE_SNK_Discovery** state the *Sink Policy Engine* waits for *VBUS* to be present.

The *Policy Engine Shall* transition to the **PE_SNK_Wait_for_Capabilities** state when:

- The *Device Policy Manager* indicates that *VBUS* has been detected.

8.3.3.3.3 PE_SNK_Wait_for_Capabilities State

On entry to the **PE_SNK_Wait_for_Capabilities** state the *Policy Engine Shall* initialize and start the **SinkWaitCapTimer**.

The *Policy Engine Shall* transition to the **PE_SNK_Evaluate_Capability** state when:

- The *Sink* is in *SPR Mode* and a **Source_Capabilities Message** is received or
- The *Sink* is in *EPR Mode* and an **EPR_Source_Capabilities Message** is received.

When the **SinkWaitCapTimer** times out, the *Policy Engine* will perform a *Hard Reset*.

8.3.3.3.4 PE_SNK_Evaluate_Capability State

The **PE_SNK_Evaluate_Capability** state is first entered when the *Sink* receives its first **Source_Capabilities Message** from the *Source*. At this point the *Sink* knows that it is *Attached* to and communicating with a PD capable *Source*.

On entry to the **PE_SNK_Evaluate_Capability** state the *Policy Engine Shall* request the *Device Policy Manager* to evaluate the supplied *Source Capabilities* based on *Local Policy*. The *Device Policy Manager Shall* indicate to the *Policy Engine* the new power level required, selected from the present offered capabilities. The *Device Policy Manager Shall* also indicate to the *Policy Engine* a *Capabilities Mismatch* if the offered power does not meet the device's requirements.

The *Policy Engine Shall* transition to the **PE_SNK_Select_Capability** state when:

- A response is received from the *Device Policy Manager*.

8.3.3.3.5 PE_SNK_Select_Capability State

On entry to the **PE_SNK_Select_Capability** state the *Policy Engine Shall* request the *Protocol Layer* to send a response *Message*, based on the evaluation from the *Device Policy Manager*. The *Message Shall* be one of the following:

- A Request from the offered *Source Capabilities*.

- A Request from the offered *Source Capabilities* with an indication that another power level would be preferred (*Capability Mismatch* bit set).

When in *SPR Mode* a *Request* Message **Shall** be sent.

When in *EPR Mode* an *EPR_Request* Message **Shall** be sent.

The *Policy Engine* **Shall** initialize and run the *SenderResponseTimer*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Transition_Sink* state when:

- An *Accept* Message is received from the *Source*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Wait_for_Capabilities* state when:

- There is no *Explicit Contract* in place and
- A *Reject* Message is received from the *Source* or
- A *Wait* Message is received from the *Source*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Ready* state when:

- There is an *Explicit Contract* in place and
- A *Reject* Message is received from the *Source* or
- A *Wait* Message is received from the *Source*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Hard_Reset* state when:

- A *SenderResponseTimer* timeout occurs.

8.3.3.3.6 PE_SNK_Transition_Sink State

On entry to the *PE_SNK_Transition_Sink* state the *Policy Engine* **Shall** initialize and run the *PSTransitionTimer* (timeout will lead to a *Hard Reset* see [Section 8.3.3.3.8, "PE_SNK_Hard_Reset State"](#) and **Shall** then request the *Device Policy Manager* to transition the *Sink's* power supply to the new power level.

Note: If there is no power level change the *Device Policy Manager* **Should Not** affect any change to the power supply.

On exit from the *PE_SNK_Transition_Sink* state the *Policy Engine* **Shall** request the *Device Policy Manager* to transition the *Sink's* power supply to the new power level.

The *Policy Engine* **Shall** transition to the *PE_SNK_Ready* state when:

- A *PS_RDY* Message is received from the *Source*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Hard_Reset* state when:

- A *Protocol Error* occurs.

8.3.3.3.7 PE_SNK_Ready State

In the *PE_SNK_Ready* state the *PD Sink* **Shall** be operating at a stable power level with no ongoing *Negotiation*. It **Shall** respond to requests from the *Source*, events from the *Device Policy Manager*.

On entry to the *PE_SNK_Ready* state as the result of a wait the *Policy Engine* **Should** do the following:

- Initialize and run the *SinkRequestTimer*.

On entry to the *PE_SNK_Ready* state if this is a *VCONN Source* which needs to establish communication with a *Cable Plug*, then the *Policy Engine* **Shall** do the following:

- Initialize and run the *DiscoverIdentityTimer* (no *GoodCRC* Message response yet received to *Discover Identity* Message).

On entry to the **PE_SNK_Ready** state if the current *Explicit Contract* is for an *SPR PPS APDO*, then the *Policy Engine* **Shall** do the following:

- Initialize and run the **SinkPPSPeriodicTimer**.

On entry to the **PE_SNK_Ready** state if the *Sink* supports *Fast Role Swap*, then the *Policy Engine* **Shall** do the following:

- Send a **Get_Sink_Cap** Message.

On exit from the **PE_SNK_Ready** state, if the transition is as a result of a DPM request to start a new *Atomic Message Sequence* (AMS) then the *Policy Engine* **Shall** notify the *Protocol Layer* that the first *Message* in an AMS will follow.

The *Policy Engine* **Shall** transition to the **PE_SNK_Evaluate_Capability** state when:

- In *SPR Mode* and a **Source_Capabilities** Message is received or
- In *EPR Mode* and an **EPR_Source_Capabilities** Message is received.

The *Policy Engine* **Shall** transition to the **PE_SNK_Select_Capability** state when:

- A new power level is requested by the *Device Policy Manager* or
- A **SinkRequestTimer** timeout occurs or
- A **SinkPPSPeriodicTimer** timeout occurs.

The *Policy Engine* **Shall** transition to the **PE_SNK_Give_Sink_Cap** state when:

- **Get_Sink_Cap** Message is received or
- **EPR_Get_Sink_Cap** Message is received.

The *Policy Engine* **Shall** transition to the **PE_SNK_Get_Source_Cap** state when:

- The *Device Policy Manager* requests an update of the remote *Source Capabilities*.

The *Policy Engine* **Shall** transition to the **PE_SNK_EPR_Keep_Alive** state when:

- The **SinkEPRKeepAliveTimer** timeouts out.

8.3.3.3.8 PE_SNK_Hard_Reset State

The *Policy Engine* **Shall** transition to the **PE_SNK_Hard_Reset** state from any state when:

- (**PSTransitionTimer** times out) and
- (**HardResetCounter** ≤ **nHardResetCount**) |
- *Hard Reset* request from *Device Policy Manager* or
- In *EPR Mode* and
 - An **EPR_Source_Capabilities** Message is received with an *EPR (A)PDO* in object positions 1...7 or
 - A **Source_Capabilities** Message is received that has not been requested using a **Get_Source_Cap** Message.

The *Policy Engine* **May** transition to the **PE_SNK_Hard_Reset** state from any state when:

- **SinkWaitCapTimer** times out

Note: If the **SinkWaitCapTimer** times out and the **HardResetCounter** is greater than **nHardResetCount** the *Sink* **Shall** assume that the *Source* is non-responsive.

Note: The **HardResetCounter** is reset on a power cycle or *Detach*.

On entry to the **PE_SNK_Hard_Reset** state the *Policy Engine* **Shall** request the generation of **Hard Reset Signaling** by the PHY Layer and increment the **HardResetCounter**.

The *Policy Engine* **Shall** transition to the **PE_SNK_Transition_to_default** state when:

- The *Hard Reset* is complete.

8.3.3.3.9 PE_SNK_Transition_to_default State

The *Policy Engine* **Shall** transition from any state to **PE_SNK_Transition_to_default** state when:

- **Hard Reset Signaling** is detected.

When **Hard Reset Signaling** is received or transmitted then the *Policy Engine* **Shall** transition from any state to **PE_SNK_Transition_to_default**. This state can also be entered from the **PE_SNK_Hard_Reset** state.

On entry to the **PE_SNK_Transition_to_default** state the *Policy Engine* **Shall**:

- indicate to the *Device Policy Manager* that the *Sink* **Shall** transition to default
- request a reset of the local hardware
- request the *Device Policy Manager* that the **Port Data Role** is set to *UFP*.

The *Policy Engine* **Shall** transition to the **PE_SNK_Startup** state when:

- The *Device Policy Manager* indicates that the *Sink* has reached the default level.

8.3.3.3.10 PE_SNK_Give_Sink_Cap State

- On entry to the **PE_SNK_Give_Sink_Cap** state the *Policy Engine* **Shall** request the *Device Policy Manager* for the current system capabilities. The *Policy Engine* **Shall** then request the *Protocol Layer* to send a **Sink_Capabilities** Message containing these capabilities. The *Policy Engine* **Shall** send:
 - A **Sink_Capabilities** Message when a **Get_Sink_Cap** Message is received or
 - An **EPR_Sink_Capabilities** Message when a **EPR_Get_Sink_Cap** Message is received.

The *Policy Engine* **Shall** transition to the **PE_SNK_Ready** state when:

- The **Sink_Capabilities** Message has been successfully sent.

8.3.3.3.11 PE_SNK_EPR_Keep_Alive

On entry to the **PE_SNK_EPR_Keep_Alive** State the *Policy Engine* **Shall** send an **EPR_KeepAlive** Message and initialize and run the **SenderResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_SNK_Ready** state when:

- A **EPR_KeepAlive_Ack** Message is received.

The *Policy Engine* **Shall** transition to the **PE_SNK_Hard_Reset** state when:

- The **SenderResponseTimer** times out.

8.3.3.3.12 PE_SNK_Get_Source_Cap State

- On entry to the **PE_SNK_Get_Source_Cap** state the *Policy Engine* **Shall** request the *Protocol Layer* to send a **get Source Capabilities** Message in order to retrieve the *Source Capabilities*. The *Policy Engine* **Shall** send:
 - A **Get_Source_Cap** Message when the *Device Policy Manager* requests *SPR* capabilities or
 - An **EPR_Get_Source_Cap** Message when the *Device Policy Manager* requests *EPR Capabilities*.

The *Policy Engine* **Shall** then start the **SenderResponseTimer**.

On exit from the **PE_SNK_Get_Source_Cap** State the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

The *Policy Engine* **Shall** transition to the **PE_SNK_Ready** state when:

- In *EPR Mode* and *SPR Source Capabilities* were requested and a **Source_Capabilities** Message is received or
- In *SPR Mode* and *EPR Source Capabilities* were requested and an **EPR_Source_Capabilities** Message is received or
- The **SenderResponseTimer** times out.

The *Policy Engine* **Shall** transition to the **PE_SNK_Evaluate_Capability** State when:

- In *SPR Mode* and *SPR Source Capabilities* were requested and a **Source_Capabilities** Message is received or
- In *EPR Mode* and *EPR Source Capabilities* were requested and an **EPR_Source_Capabilities** Message is received.

8.3.3.4

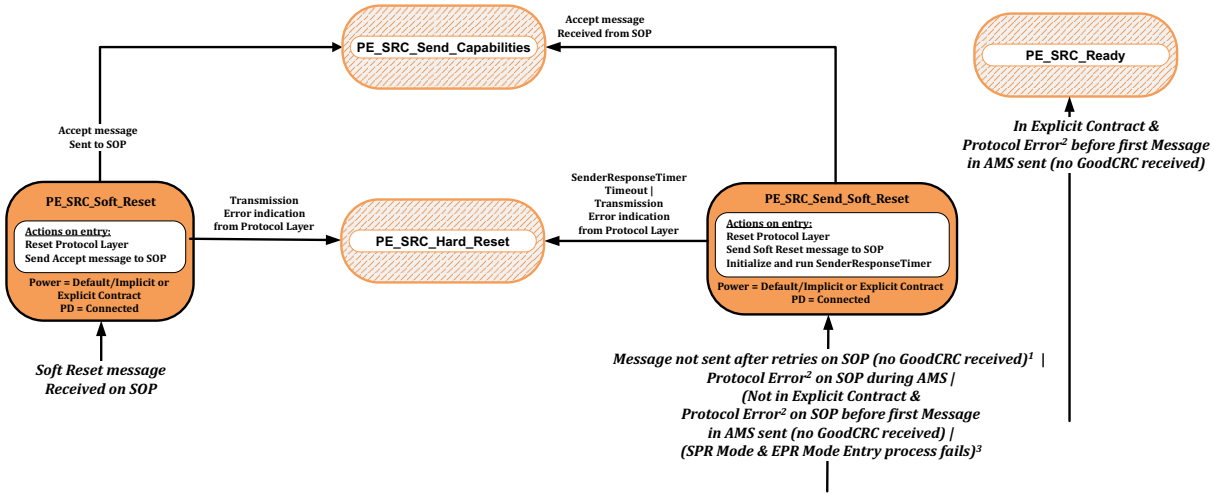
SOP Soft Reset and Protocol Error State Diagrams

8.3.3.4.1

SOP Source Port Soft Reset and Protocol Error State Diagram

Figure 8.134, "SOP Source Port Soft Reset and Protocol Error State Diagram" below shows the state diagram for the Policy Engine in a Source Port when performing a Soft Reset of its Port Partner i.e., using SOP. The following sections describe operation in each of the states.

Figure 8.134 SOP Source Port Soft Reset and Protocol Error State Diagram



¹⁾ Excludes the *Soft_Reset* Message itself.

²⁾ An Unrecognized or Unsupported Message received on *SOP* will result in a *Not_Supported* Message response being generated on *SOP* (see Section 6.3.16 "Not_Supported Message").

³⁾ See Section 6.4.10.1 "Process to enter EPR Mode" for the conditions when a *Soft_Reset* Message *Shall* be sent by the Source during the EPR Mode entry process.

8.3.3.4.1.1

PE_SRC_Send_Soft_Reset State

The **PE_SRC_Send_Soft_Reset** state *Shall* be entered from any state when:

- A Protocol Error on SOP is detected by the Protocol Layer during a Non-interruptible AMS (see Section 6.8.1, "Soft Reset and Protocol Error") or
- A Message has not been sent after retries to the Sink or
- When not in an Explicit Contract and Protocol Errors occurred on SOP during any AMS where the first Message in the AMS has not yet been sent i.e., an unexpected Message is received instead of the expected **GoodCRC** Message response or
- When in SPR Mode and the EPR Mode entry process fails.

The main exceptions to this rule are when:

- The Source is in the **PE_SRC_Send_Capabilities** state, there is a **Source_Capabilities** Message sending failure on SOP (without a **GoodCRC** Message) and the Source is not presently Attached (as indicated in Figure 8.132, "Source Port State Diagram"). In this case, the **PE_SRC_Discovery** state is entered (see Section 8.3.3.2.2, "PE_SRC_Discovery State").
- When the voltage is in transition due to a new Explicit Contract being Negotiated (see Section 8.3.3.2, "Policy Engine Source Port State Diagram"). In this case **Hard Reset** Signaling will be generated.
- During a Power Role Swap when the power supply is in transition (see Section 8.3.3.19.3, "Policy Engine in Source to Sink Power Role Swap State Diagram" and Section 8.3.3.19.4, "Policy Engine in Sink to Source Power Role Swap State Diagram"). In this case USB Type-C Error Recovery will be triggered directly.
- During a Data Role Swap when there is a mismatch in the **Port Data Role** field (see Section 6.2.1.1.6, "Port Data Role"). In this case USB Type-C Error Recovery will be triggered directly.

Protocol Errors occurring in the following situations **Shall Not** lead to a *Soft Reset*, but **Shall** result in a transition to the **PE_SRC_Ready** state where the *Message* received will be handled as if it had been received in the **PE_SRC_Ready** state:

- When in an *Explicit Contract* and *Protocol Errors* occurred on *SOP* during any *AMS* where the first *Message* in the *AMS* has not yet been sent i.e., an unexpected *Message* is received instead of the expected **GoodCRC** *Message* response.

On entry to the **PE_SRC_Send_Soft_Reset** state the *Policy Engine* **Shall** request the *SOP Protocol Layer* to perform a *Soft Reset*, then **Shall** send a **Soft_Reset** *Message* to the *Sink* on *SOP*, and initialize and run the **SenderResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_SRC_Send_Capabilities** state when:

- An **Accept** *Message* has been received on *SOP*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- A **SenderResponseTimer** timeout occurs.
- Or the *Protocol Layer* indicates that a transmission error has occurred.

8.3.3.4.1.2 PE_SRC_Soft_Reset State

The **PE_SRC_Soft_Reset** state **Shall** be entered from any state when a **Soft_Reset** *Message* is received on *SOP* from the *Protocol Layer*.

On entry to the **PE_SRC_Soft_Reset** state the *Policy Engine* **Shall** reset the *SOP Protocol Layer* and **Shall** then request the *Protocol Layer* to send an **Accept** *Message* on *SOP*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Send_Capabilities** state (see [Section 8.3.3.2.3. "PE_SRC_Send_Capabilities State"](#)) when:

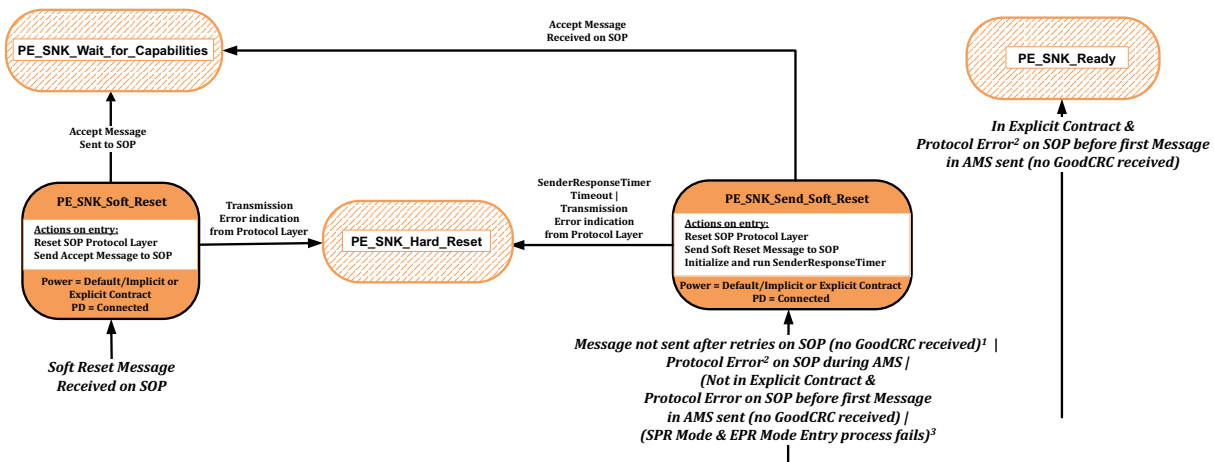
- The **Accept** *Message* has been sent on *SOP*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- The *Protocol Layer* indicates that a transmission error has occurred.

Figure 8.135, "Sink Port Soft Reset and Protocol Error Diagram" below shows the state diagram for the Policy Engine in a Sink Port when performing a Soft Reset of its Port Partner i.e., using SOP. The following sections describe operation in each of the states.

Figure 8.135 Sink Port Soft Reset and Protocol Error Diagram



¹⁾ Excludes the *Soft_Reset* Message itself.

²⁾ An Unrecognized or Unsupported Message will result in a *Not_Supported* Message response being generated (see Section 6.3.16 "Not_Supported Message").

³⁾ See Section 6.4.10.1 "Process to enter EPR Mode" for the conditions when a *Soft_Reset* Message **shall** be sent by the Sink during the EPR Mode entry process.

8.3.3.4.2.1

PE_SNK_Send_Soft_Reset State

The **PE_SNK_Send_Soft_Reset** state **shall** be entered from any state when:

- A Protocol Error on SOP is detected by the Protocol Layer during an AMS (see Section 6.8.1, "Soft Reset and Protocol Error") or
- A Message has not been sent after retries to the Sink or
- When not in an Explicit Contract and Protocol Errors occurred on SOP during any AMS where the first Message in the AMS has not yet been sent i.e., an unexpected Message is received instead of the expected **GoodCRC** Message response.
- When in SPR Mode and the EPR Mode entry process fails.

The main exceptions to this rule are when:

- When the voltage is in transition due to a new Explicit Contract being Negotiated (see Section 8.3.3.3, "Policy Engine Sink Port State Diagram"). In this case a Hard Reset will be generated.
- During a Power Role Swap when the power supply is in transition (see Section 8.3.3.19.3, "Policy Engine in Source to Sink Power Role Swap State Diagram" and Section 8.3.3.19.4, "Policy Engine in Sink to Source Power Role Swap State Diagram"). In this case a Hard Reset will be triggered directly.
- During a Data Role Swap when the DFP/UFP Data Roles are changing. In this case USB Type-C Error Recovery will be triggered directly.

Note: Protocol Errors occurring in the following situations **shall Not** lead to a Soft Reset, but **shall** result in a transition to the **PE_SNK_Ready** state where the Message received will be handled as if it had been received in the **PE_SNK_Ready** state:

- When in an Explicit Contract and Protocol Errors occurred on SOP during any AMS where the first Message in the AMS has not yet been sent i.e., an unexpected Message is received instead of the expected **GoodCRC** Message response.

On entry to the **PE_SNK_Send_Soft_Reset** state the *Policy Engine* **Shall** request the *SOP Protocol Layer* to perform a *Soft Reset*, then **Shall** send a **Soft_Reset** Message on *SOP* to the *Source*, and initialize and run the **SenderResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_SNK_Wait_for_Capabilities** state when:

- An **Accept** Message has been received on *SOP*.

The *Policy Engine* **Shall** transition to the **PE_SNK_Hard_Reset** state when:

- A **SenderResponseTimer** timeout occurs.
- Or the *Protocol Layer* indicates that a transmission error has occurred.

8.3.3.4.2.2 PE_SNK_Soft_Reset State

The **PE_SNK_Soft_Reset** state **Shall** be entered from any state when a **Soft_Reset** Message is received on *SOP* from the *Protocol Layer*.

On entry to the **PE_SNK_Soft_Reset** state the *Policy Engine* **Shall** reset the *SOP Protocol Layer* and **Shall** then request the *Protocol Layer* to send an **Accept** Message on *SOP*.

The *Policy Engine* **Shall** transition to the **PE_SNK_Wait_for_Capabilities** state when:

- The **Accept** Message has been sent on *SOP*.

The *Policy Engine* **Shall** transition to the **PE_SNK_Hard_Reset** state when:

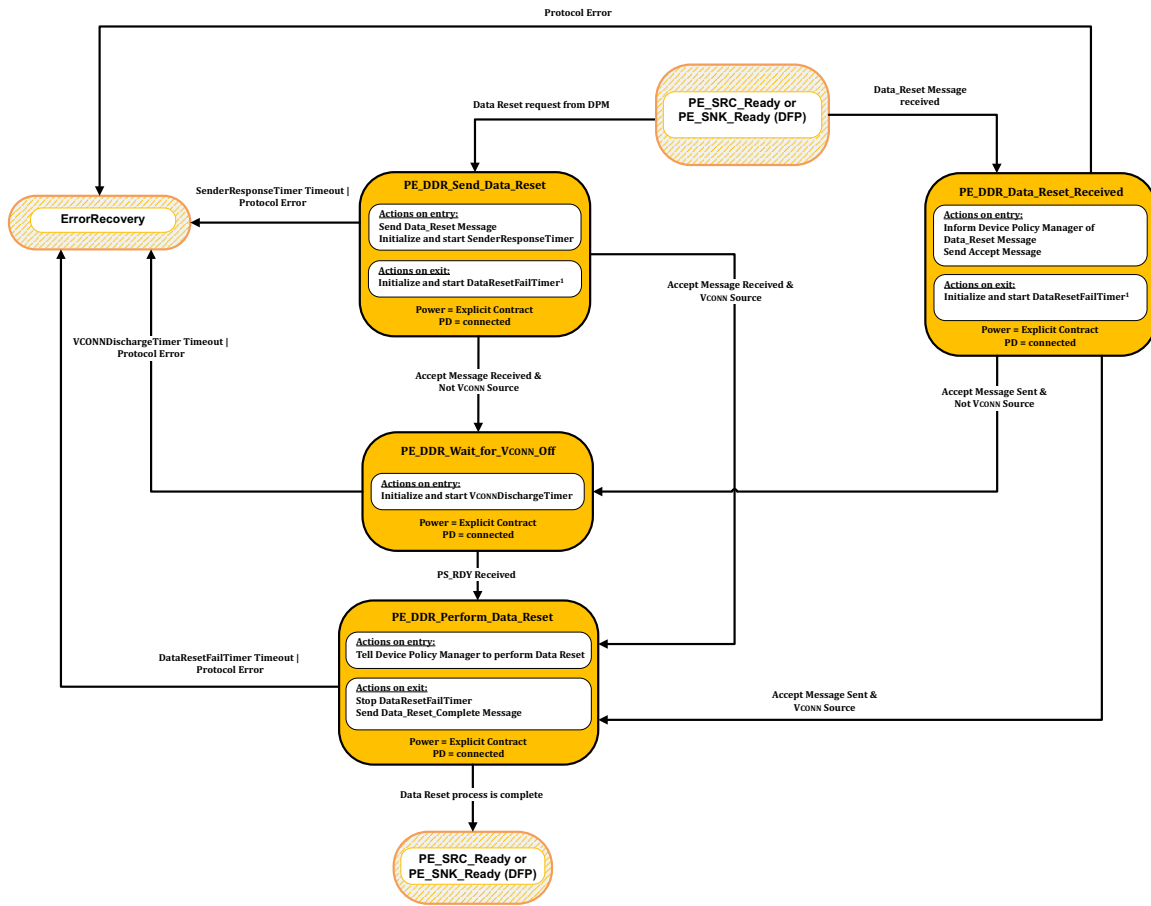
- The *Protocol Layer* indicates that a transmission error has occurred.

8.3.3.5 Data Reset State Diagrams

8.3.3.5.1 DFP Data_Reset Message State Diagrams

Figure 8.136, "DFP Data_Reset Message State Diagram" shows the state diagram for a *Data_Reset* Message sent or received by a DFP.

Figure 8.136 DFP Data_Reset Message State Diagram



¹⁾ Note that the *DataResetFailTimer* **Shall** continue to run in every state until it is stopped or times out.

8.3.3.5.1.1 PE_DDR_Send_Data_Reset State

The **PE_DDR_Send_Data_Reset** State **Shall** be entered from the **PE_SRC_Ready** or **PE_SNK_Ready** State when requested by the *Device Policy Manager*.

On entry to the **PE_DDR_Send_Data_Reset** State the *Policy Engine* **Shall** request the *Protocol Layer* to send a *Data_Reset* Message and then initialize and start the *SenderResponseTimer*.

On exit from the **PE_DDR_Send_Data_Reset** State the *Policy Engine* **Shall** initialize and start the *DataResetFailTimer*.

The *Policy Engine* **Shall** transition to the **PE_DDR_Perform_Data_Reset** State when:

- An *Accept* Message has been received and
- The DFP is presently the *VCONN Source*.

The *Policy Engine* **Shall** transition to the **PE_DDR_Wait_For_VCONN_Off** State when:

- An *Accept* Message has been received and

- The *DFP* is not presently the *VCONN Source*.

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- A *SenderResponseTimer* timeout occurs or
- A *Protocol Error* occurs.

8.3.3.5.1.2 PE_DDR_Data_Reset_Received State

The *PE_DDR_Data_Reset_Received* State **Shall** be entered from the *PE_SRC_Ready* or *PE_SNK_Ready* State when a *Data_Reset* Message is received.

On entry to the *PE_DDR_Data_Reset_Received* State the *Policy Engine* **Shall** inform the *Device Policy Manager* and then **Shall** send an *Accept* Message.

On exit from the *PE_DDR_Data_Reset_Received* State the *Policy Engine* **Shall** initialize and start the *DataResetFailTimer*.

The *Policy Engine* **Shall** transition to the *PE_DDR_Perform_Data_Reset* State when:

- An *Accept* Message has been sent and
- The *DFP* is presently the *VCONN Source*.

The *Policy Engine* **Shall** transition to the *PE_DDR_Wait_For_VCONN_Off* State when:

- An *Accept* Message has been sent and
- The *DFP* is not presently the *VCONN Source*.

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- A *Protocol Error* occurs.

8.3.3.5.1.3 PE_DDR_Wait_For_VCONN_Off State

On entry to the *PE_DDR_Wait_For_VCONN_Off* State the *Policy Engine* **Shall** initialize and start the *VCONNDischargeTimer*.

The *Policy Engine* **Shall** transition to the *PE_DDR_Perform_Data_Reset* State when:

- A *PS_RDY* Message is received.

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- The *VCONNDischargeTimer* has timed out or
- A *Protocol Error* occurs.

8.3.3.5.1.4 PE_DDR_Perform_Data_Reset State

On entry to the *PE_DDR_Perform_Data_Reset* State the *Policy Engine* **Shall** request the *Device Policy Manager* to complete the *Data Reset* process as defined in [Section 6.3.14, "Data_Reset Message"](#).

On exit from the *PE_DDR_Perform_Data_Reset* State the *Policy Engine* **Shall** stop the *DataResetFailTimer* and send a *Data_Reset_Complete* Message.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready* or *PE_SNK_Ready* State depending on the *DFP*'s *Power Role* when:

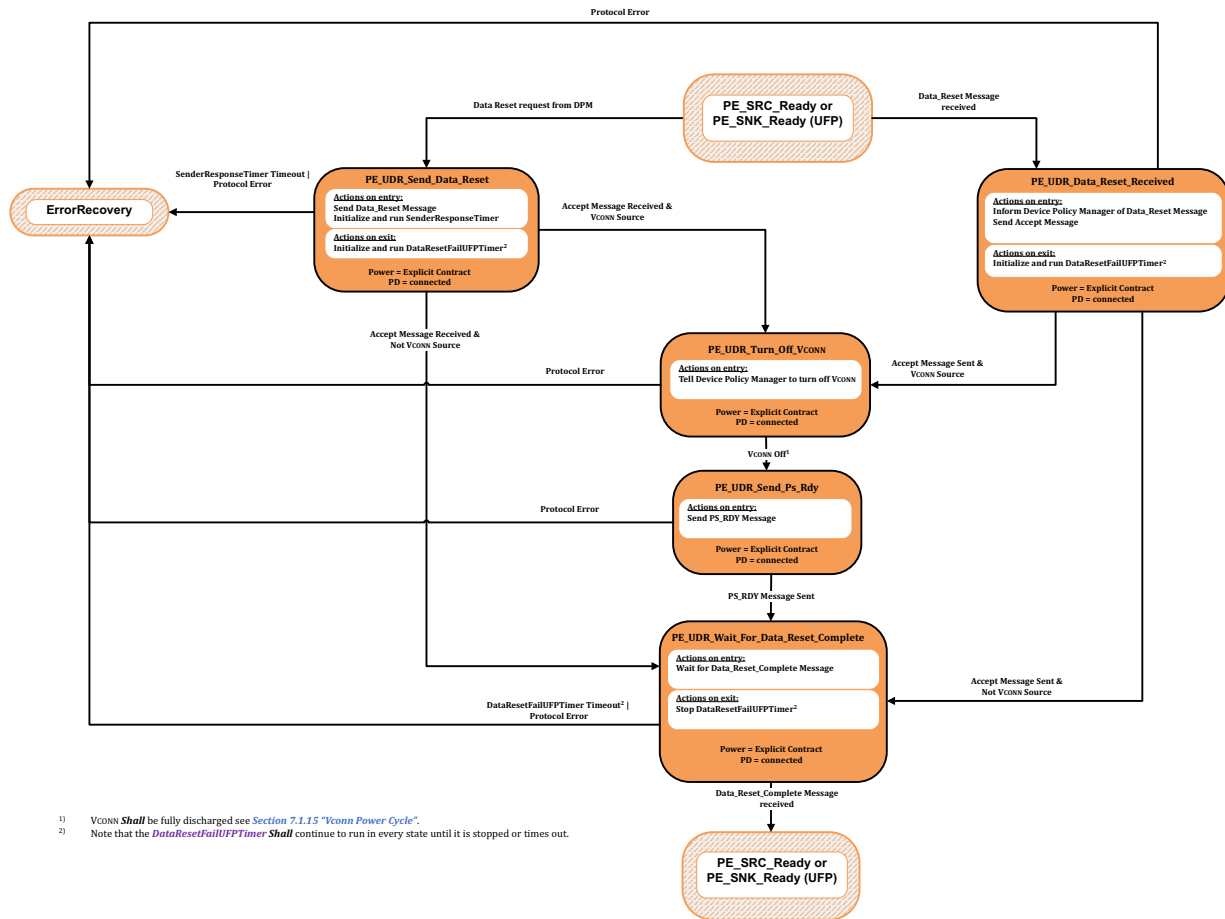
- The *DPM* indicates that *Data Reset* process is complete (see [Section 6.3.14, "Data_Reset Message"](#)).

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- The *DataResetFailTimer* times out
- A *Protocol Error* occurs.

Figure 8.137, "UFP Data_Reset Message State Diagram" shows the state diagram for a *Data_Reset* Message sent or received by a UFP.

Figure 8.137 UFP Data_Reset Message State Diagram



8.3.3.5.2.1

PE_UDR_Send_Data_Reset State

The *PE_UDR_Send_Data_Reset* State **shall** be entered from the *PE_SRC_Ready* or *PE_SNK_Ready* State when requested by the *Device Policy Manager*.

On entry to the *PE_UDR_Send_Data_Reset* State the *Policy Engine* **shall** request the *Protocol Layer* to send a *Data_Reset* Message and then initialize and run the *SenderResponseTimer*.

On exit from the *PE_UDR_Send_Data_Reset* State the *Policy Engine* **shall** initialize and run the *DataResetFailUFPTimer*.

The *Policy Engine* **shall** transition to the *PE_UDR_Turn_Off_VCONN* State when:

- An *Accept* Message has been received and
- The UFP is presently the *VCONN* Source.

The *Policy Engine* **shall** transition to the *PE_UDR_Wait_For_Data_Reset_Complete* State when:

- An *Accept* Message has been received and
- The UFP is not presently the *VCONN* Source.

The *Policy Engine* **shall** transition to *ErrorRecovery* when:

- The *SenderResponseTimer* has timed out or
- A *Protocol Error* occurs.

8.3.3.5.2.2 PE_UDR_Data_Reset_Received State

The *PE_UDR_Data_Reset_Received* State **Shall** be entered from either the *PE_SRC_Ready* or *PE_SNK_Ready* State when a *Data_Reset* Message is received.

On entry to the *PE_UDR_Data_Reset_Received* State the *Policy Engine* **Shall** inform the *Device Policy Manager* and then **Shall** send an *Accept* Message.

On exit from the *PE_UDR_Data_Reset_Received* State the *Policy Engine* **Shall** initialize and run the *DataResetFailUFPTimer*.

The *Policy Engine* **Shall** transition to the *PE_UDR_Turn_Off_VCONN* State when:

- An *Accept* Message has been sent and
- The *UFP* is presently the *VCONN Source*.

The *Policy Engine* **Shall** transition to the *PE_UDR_Wait_For_Data_Reset_Complete* State when:

- An *Accept* Message has been sent and
- The *UFP* is not presently the *VCONN Source*.

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- A *Protocol Error* occurs.

8.3.3.5.2.3 PE_UDR_Turn_Off_VCONN State

On entry to the *PE_UDR_Turn_Off_VCONN* State the *Policy Engine* **Shall** request the *Device Policy Manager* to turn off *VCONN*.

The *Policy Engine* **Shall** transition to the *PE_UDR_Send_Ps_Rdy* State when:

- The *DPM* indicates that *VCONN* has been turned off (*VCONN* below *vRaReconnect* see [USB Type-C 2.4]).

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- A *Protocol Error* occurs.

8.3.3.5.2.4 PE_UDR_Send_Ps_Rdy State

On entry to the *PE_UDR_Send_Ps_Rdy* State the *Policy Engine* **Shall** send a *PS_RDY* Message.

The *Policy Engine* **Shall** transition to the *PE_UDR_Wait_For_Data_Reset_Complete* State when:

- The *PS_RDY* Message has been sent.

The *Policy Engine* **Shall** transition to *ErrorRecovery* when:

- A *Protocol Error* occurs.

8.3.3.5.2.5 PE_UDR_Wait_For_Data_Reset_Complete State

On entry to the *PE_UDR_Wait_For_Data_Reset_Complete* State the *Policy Engine* **Shall** wait for the *Data_Reset_Complete* Message.

On exit from the *PE_UDR_Wait_For_Data_Reset_Complete* State the *Policy Engine* **Shall** stop the *DataResetFailUFPTimer*.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready* or *PE_SNK_Ready* State depending on the *UFP's* *Power Role* when:

- The *Data_Reset_Complete* Message is received.

The Policy Engine **Shall** transition to *ErrorRecovery* when:

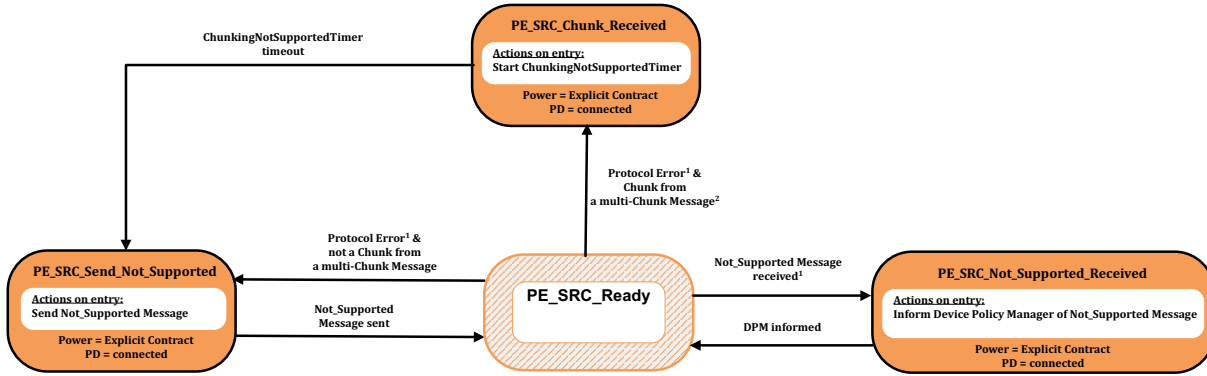
- The *DataResetFailUFPTimer* times out or
- A *Protocol Error* occurs.

8.3.3.6 Not Supported Message State Diagrams

8.3.3.6.1 Source Port Not Supported Message State Diagram

Figure 8.138, "Source Port Not Supported Message State Diagram" shows the state diagram for a *Not_Supported* Message sent or received by a Source Port.

Figure 8.138 Source Port Not Supported Message State Diagram



- 1) Transition as a result of an unsupported Message being received in the *PE_SRC_Ready* state directly (see also [Section 8.3.3.4.1 "SOP Source Port Soft Reset and Protocol Error State Diagram"](#)).
- 2) Transition can only occur where a manufacturer has opted not to implement a Chunking state machine (see [Section 6.12.2.1 "Protocol Layer Chunking"](#)) and is communicating with a system which is attempting to send it Chunks.

8.3.3.6.1.1 PE_SRC_Send_Not_Supported State

The *PE_SRC_Send_Not_Supported* state **Shall** be entered from the *PE_SRC_Ready* state either as the result of a *Protocol Error* received during an interruptible *AMS* or as a result of an *Unsupported Message* being received in the *PE_SRC_Ready* state directly except for the first *Chunk* in a multi-*Chunk Message* (see also [Section 6.12.2.1, "Protocol Layer Chunking"](#) and [Section 8.3.3.4.1, "SOP Source Port Soft Reset and Protocol Error State Diagram"](#)).

On entry to the *PE_SRC_Send_Not_Supported* state (from the *PE_SRC_Ready* state) the *Policy Engine* **Shall** request the *Protocol Layer* to send a *Not_Supported Message*.

The *Policy Engine* **Shall** transition back to the previous state (*PE_SRC_Ready* see [Figure 8.132, "Source Port State Diagram"](#)) when:

- The *Not_Supported Message* has been successfully sent.

8.3.3.6.1.2 PE_SRC_Not_Supported_Received State

The *PE_SRC_Not_Supported_Received* state **Shall** be entered from the *PE_SRC_Ready* state when a *Not_Supported Message* is received.

On entry to the *PE_SRC_Not_Supported_Received* state the *Policy Engine* **Shall** inform the *Device Policy Manager*.

The *Policy Engine* **Shall** transition back to the previous state (*PE_SRC_Ready* see [Figure 8.132, "Source Port State Diagram"](#)) when:

- The *Device Policy Manager* has been informed.

8.3.3.6.1.3 PE_SRC_Chunk_Received State

The *PE_SRC_Chunk_Received* state **Shall** be entered from the *PE_SRC_Ready* state as a result of an *Unsupported Message* being received in the *PE_SRC_Ready* state directly where the *Message* is a *Chunk* in a multi-*Chunk Message* (see also [Section 6.12.2.1, "Protocol Layer Chunking"](#) and [Section 8.3.3.4.1, "SOP Source Port Soft Reset and Protocol Error State Diagram"](#)).

On entry to the *PE_SRC_Chunk_Received* state (from the *PE_SRC_Ready* state) the *Policy Engine* **Shall** initialize and run the *ChunkingNotSupportedTimer*.

The *Policy Engine* **Shall** transition to *PE_SRC_Send_Not_Supported* when:

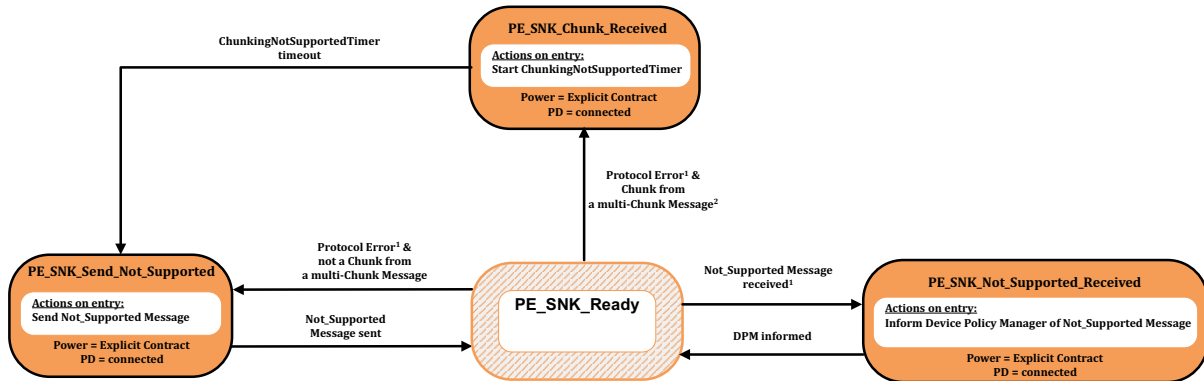
- The *ChunkingNotSupportedTimer* has timed out.

8.3.3.6.2

Sink Port Not Supported Message State Diagram

Figure 8.139, "Sink Port Not Supported Message State Diagram" shows the state diagram for a *Not_Supported* Message sent or received by a *Sink Port*.

Figure 8.139 Sink Port Not Supported Message State Diagram



- 1) Transition as a result of an unsupported Message being received in the *PE_SNK_Ready* state directly (see also Section 8.3.3.4.2 "SOP Sink Port Soft Reset and Protocol Error State Diagram").
- 2) Transition can only occur where a manufacturer has opted not to implement a Chunking state machine (see Section 6.12.2.1 "Protocol Layer Chunking") and is communicating with a system which is attempting to send it Chunks.

8.3.3.6.2.1

PE_SNK_Send_Not_Supported State

The *PE_SNK_Send_Not_Supported* state **shall** be entered from the *PE_SNK_Ready* state either as the result of a *Protocol Error* received during an interruptible *AMS* or as a result of an *Unsupported Message* being received in the *PE_SNK_Ready* state directly except for the first *Chunk* in a multi-*Chunk Message* (see also Section 6.12.2.1, "Protocol Layer Chunking" and Section 8.3.3.4.1, "SOP Source Port Soft Reset and Protocol Error State Diagram").

On entry to the *PE_SNK_Send_Not_Supported* state (from the *PE_SNK_Ready* state) the *Policy Engine* **shall** request the *Protocol Layer* to send a *Not_Supported* Message.

The *Policy Engine* **shall** transition back to the previous state (*PE_SNK_Ready* see Figure 8.133, "Sink Port State Diagram") when:

- The *Not_Supported* Message has been successfully sent.

8.3.3.6.2.2

PE_SNK_Not_Supported_Received State

The *PE_SNK_Not_Supported_Received* state **shall** be entered from the *PE_SNK_Ready* state when a *Not_Supported* Message is received.

On entry to the *PE_SNK_Not_Supported_Received* state the *Policy Engine* **shall** inform the *Device Policy Manager*.

The *Policy Engine* **shall** transition back to the previous state (*PE_SNK_Ready* see Figure 8.133, "Sink Port State Diagram") when:

- The *Device Policy Manager* has been informed.

8.3.3.6.2.3

PE_SNK_Chunk_Received State

The *PE_SNK_Chunk_Received* state **shall** be entered from the *PE_SNK_Ready* state as a result of an *Unsupported Message* being received in the *PE_SNK_Ready* state directly where the *Message* is a *Chunk* in a multi-*Chunk Message* (see also Section 6.12.2.1, "Protocol Layer Chunking" and Section 8.3.3.4.1, "SOP Source Port Soft Reset and Protocol Error State Diagram").

On entry to the *PE_SNK_Chunk_Received* state (from the *PE_SNK_Ready* state) the *Policy Engine* **shall** initialize and run the *ChunkingNotSupportedTimer*.

The *Policy Engine* **shall** transition to *PE_SNK_Send_Not_Supported* when:

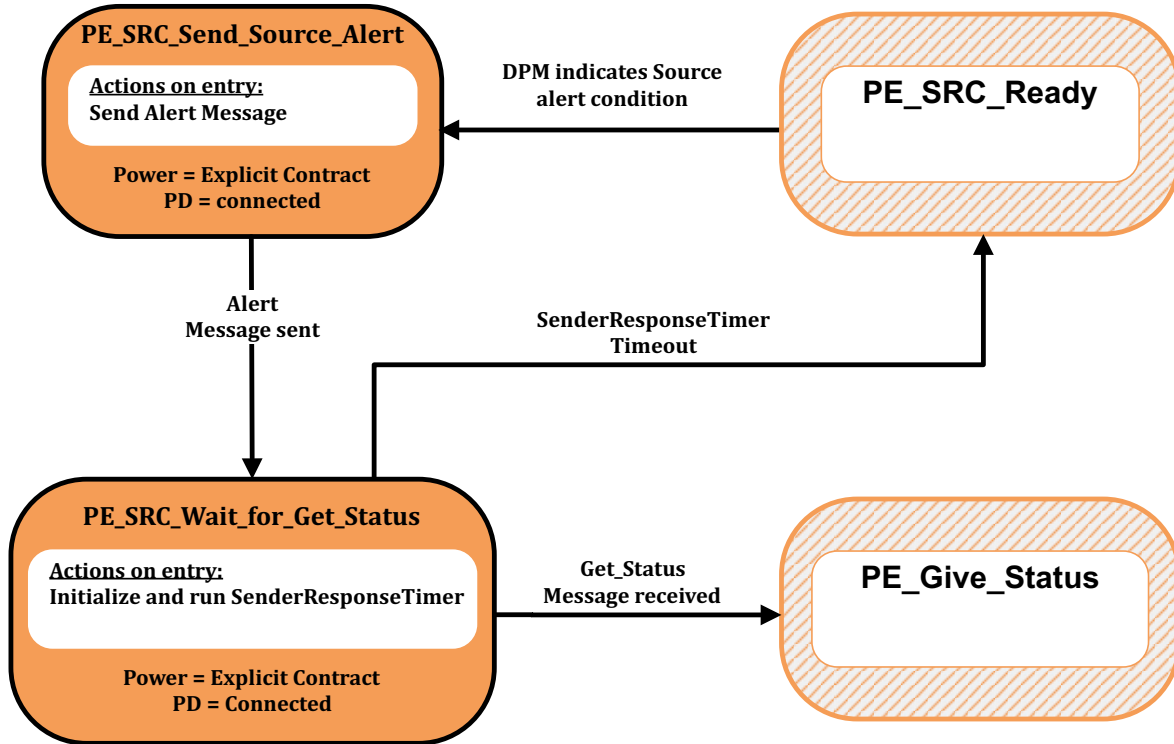
- The *ChunkingNotSupportedTimer* has timed out.

8.3.3.7 Alert State Diagrams

8.3.3.7.1 Source Port Source Alert State Diagram

[Figure 8.140, "Source Port Source Alert State Diagram"](#) shows the state diagram for an **Alert** Message sent by a *Source Port*.

Figure 8.140 Source Port Source Alert State Diagram



8.3.3.7.1.1 PE_SRC_Send_Source_Alert State

The **PE_SRC_Send_Source_Alert** state **Shall** be entered from the **PE_SRC_Ready** state when the *Device Policy Manager* indicates that there is a *Source* alert condition to be reported.

On entry to the **PE_SRC_Send_Source_Alert** state the *Policy Engine* **Shall** request the *Protocol Layer* to send an **Alert** Message.

The *Policy Engine* **Shall** transition to the **PE_SRC_Wait_for_Get_Status** State when:

- The **Alert** Message has been successfully sent.

8.3.3.7.1.2 PE_SRC_Wait_for_Get_Status State

On entry to the **PE_SRC_Wait_for_Get_Status** State the *Policy Engine* **Shall** initialize and run the **SenderResponseTimer**.

The *Policy Engine* **Shall** transition back to the **PE_Give_Status** State (see [Figure 8.151, "Give Status State Diagram"](#)) when:

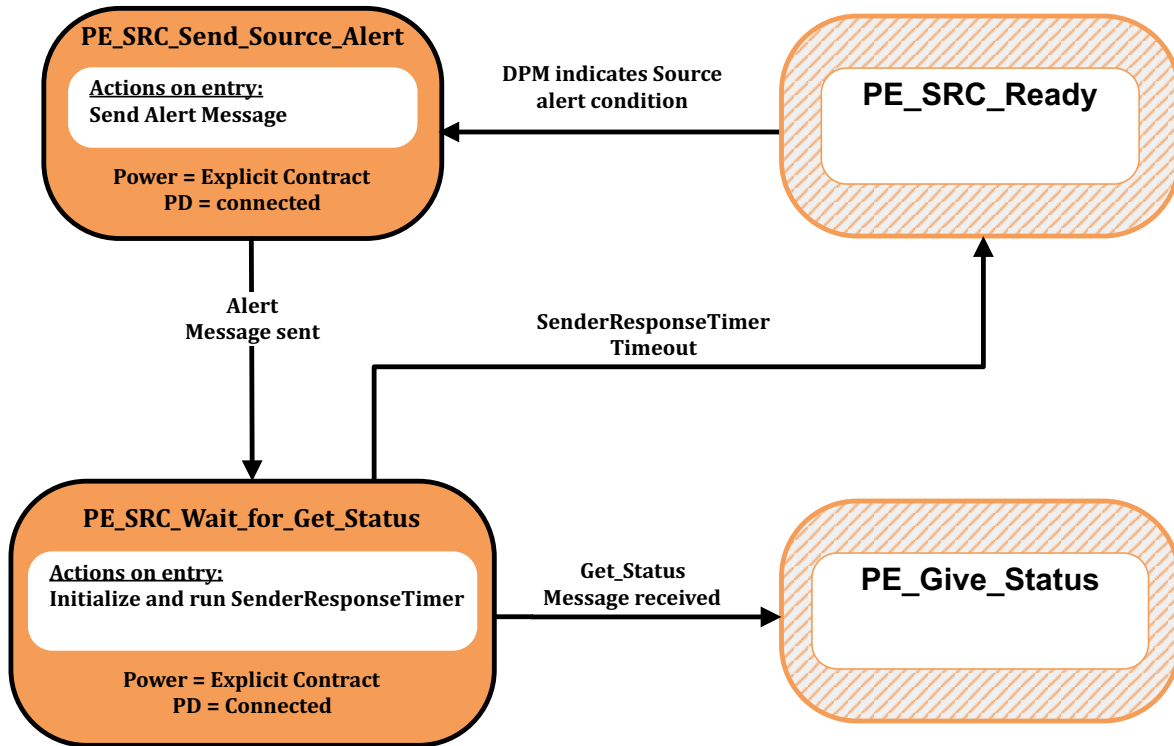
- A **Get_Status** Message is received.

The *Policy Engine* **Shall** transition back to **PE_SRC_Ready** (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- The **SenderResponseTimer** times out.

[Figure 8.141, "Sink Port Source Alert State Diagram"](#) shows the state diagram for an **Alert** Message received by a Sink Port.

Figure 8.141 Sink Port Source Alert State Diagram



8.3.3.7.2.1

PE_SNK_Source_Alert_Received State

The **PE_SNK_Source_Alert_Received** state **shall** be entered from the **PE_SNK_Ready** state when an **Alert** Message is received.

On entry to the **PE_SNK_Source_Alert_Received** state the Policy Engine **shall** inform the Device Policy Manager of the details of the Source alert.

The Policy Engine **shall** transition to the **PE_Get_Status** State (see [Figure 8.150, "Get Status State Diagram"](#)) when:

- The DPM requests status.

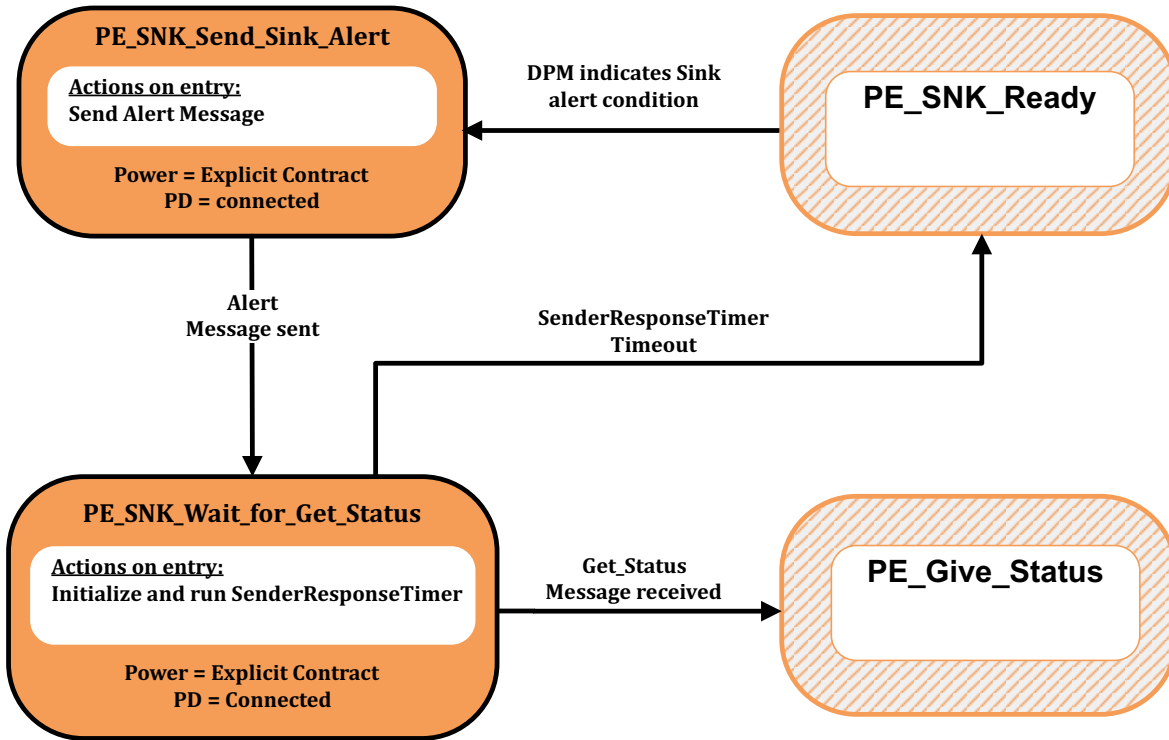
The Policy Engine **shall** transition back to the **PE_SNK_Ready** State (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- The DPM does not request status.

8.3.3.7.3 Sink Port Sink Alert State Diagram

Figure 8.142, "Sink Port Sink Alert State Diagram" shows the state diagram for an *Alert* Message sent by a Sink Port.

Figure 8.142 Sink Port Sink Alert State Diagram



8.3.3.7.3.1 PE_SNK_Send_Sink_Alert State

The **PE_SNK_Send_Sink_Alert** state **Shall** be entered from the **PE_SNK_Ready** state when the *Device Policy Manager* indicates that there is a *Source* alert condition to be reported.

On entry to the **PE_SNK_Send_Sink_Alert** state the *Policy Engine* **Shall** request the *Protocol Layer* to send an *Alert* Message.

The *Policy Engine* **Shall** transition to the **PE_SNK_Wait_for_Get_Status** State when:

- The *Alert* Message has been successfully sent.

8.3.3.7.3.2 PE_SNK_Wait_for_Get_Status State

On entry to the **PE_SNK_Wait_for_Get_Status** State the *Policy Engine* **Shall** initialize and run the *SenderResponseTimer*.

The *Policy Engine* **Shall** transition back to the **PE_Give_Status** State (see Figure 8.151, "Give Status State Diagram") when:

- A *Get_Status* Message is received.

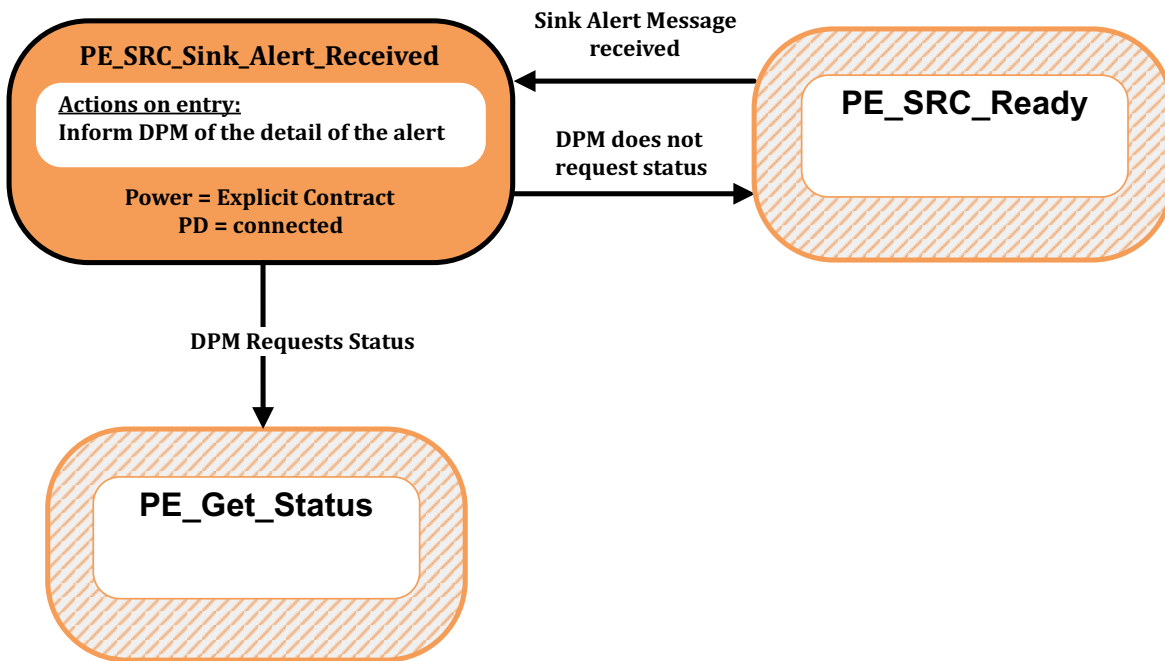
The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** (see Figure 8.133, "Sink Port State Diagram") when:

- The *SenderResponseTimer* times out.

8.3.3.7.4 Source Port Sink Alert State Diagram

[Figure 8.143, "Source Port Sink Alert State Diagram"](#) shows the state diagram for an **Alert** Message received by a *Source Port*.

Figure 8.143 Source Port Sink Alert State Diagram



8.3.3.7.4.1 PE_SRC_Sink_Alert_Received State

The **PE_SRC_Sink_Alert_Received** state **Shall** be entered from the **PE_SRC_Ready** state when an **Alert** Message is received.

On entry to the **PE_SRC_Sink_Alert_Received** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the details of the *Source* alert.

The *Policy Engine* **Shall** transition to the **PE_Get_Status** State (see [Figure 8.150, "Get Status State Diagram"](#)) when:

- The DPM requests status.

The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- The DPM does not request status.

8.3.3.8

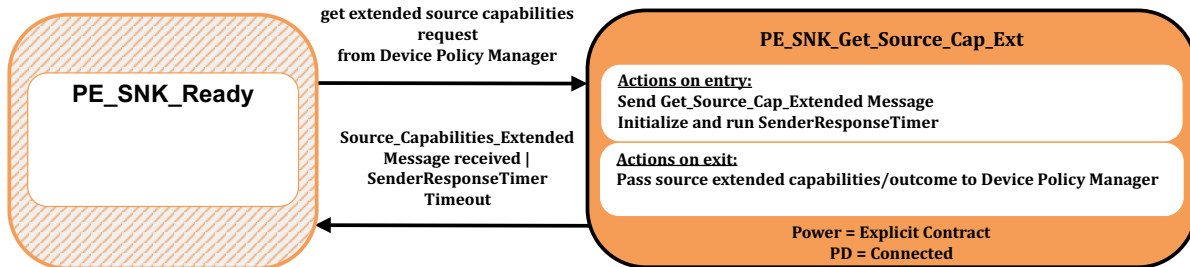
Source/Sink Capabilities Extended State Diagrams

8.3.3.8.1

Sink Port Get Source Capabilities Extended State Diagram

[Figure 8.144, "Sink Port Get Source Capabilities Extended State Diagram"](#) shows the state diagram for a *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner's* extended *Source Capabilities*. See also [Section 6.5.1, "Source Capabilities Extended Message"](#).

Figure 8.144 Sink Port Get Source Capabilities Extended State Diagram



8.3.3.8.1.1

PE_SNK_Get_Source_Cap_Ext State

The *Policy Engine* **Shall** transition to the **PE_SNK_Get_Source_Cap_Ext** state, from the **PE_SNK_Ready** state, due to a request to get the remote extended *Source Capabilities* from the *Device Policy Manager*.

On entry to the **PE_SNK_Get_Source_Cap_Ext** state the *Policy Engine* **Shall** send a **Get_Source_Cap_Extended Message** and initialize and run the **SenderResponseTimer**.

On exit from the **PE_SNK_Get_Source_Cap_Ext** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

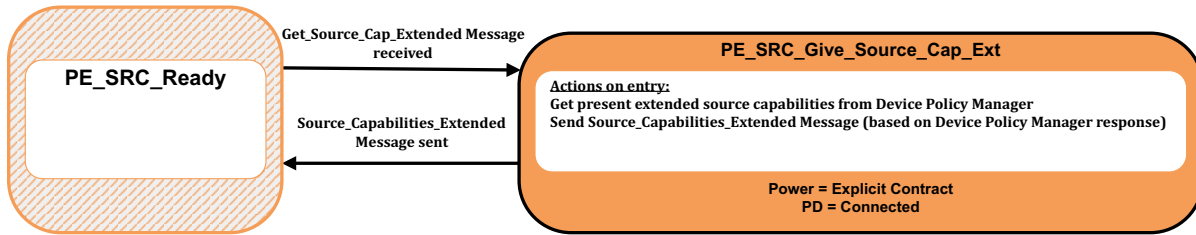
The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** state (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Source_Capabilities_Extended Message** is received
- Or **SenderResponseTimer** times out.

8.3.3.8.2 Source Give Source Capabilities Extended State Diagram

[Figure 8.145, "Source Give Source Capabilities Extended State Diagram"](#) shows the state diagram for a *Source* on receiving a *Get_Source_Cap_Extended* Message. See also [Section 6.5.1, "Source Capabilities Extended Message"](#).

Figure 8.145 Source Give Source Capabilities Extended State Diagram



8.3.3.8.2.1 PE_SRC_Give_Source_Cap_Ext State

The *Policy Engine* **Shall** transition to the *PE_SRC_Give_Source_Cap_Ext* state, from the *PE_SRC_Ready* state, when a *Get_Source_Cap_Extended* Message is received.

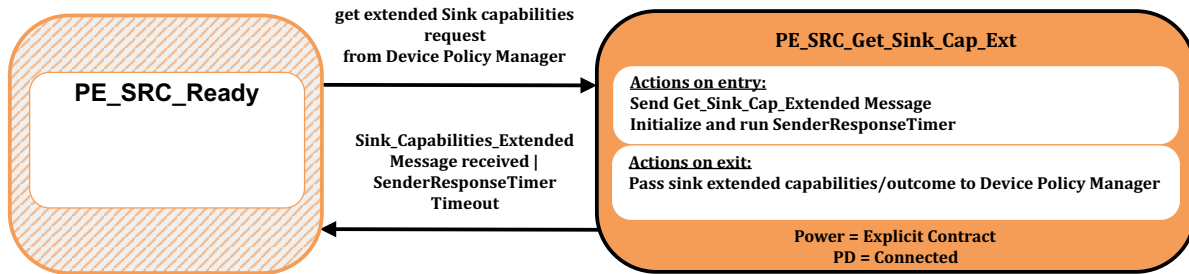
On entry to the *PE_SRC_Give_Source_Cap_Ext* state the *Policy Engine* **Shall** request the present extended *Source Capabilities* from the *Device Policy Manager* and then send a *Source_Capabilities_Extended* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to the *PE_SRC_Ready* state (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- The *Source_Capabilities_Extended* Message has been successfully sent.

Figure 8.146, "Source Port Get Sink Capabilities Extended State Diagram" shows the state diagram for a *Source* on receiving a request from the *Device Policy Manager* to get the *Port Partner's* extended *Sink Capabilities*. See also Section 6.5.13, "Sink Capabilities Extended Message".

Figure 8.146 Source Port Get Sink Capabilities Extended State Diagram



8.3.3.8.3.1

PE_SRC_Get_Sink_Cap_Ext State

The *Policy Engine* **Shall** transition to the *PE_SRC_Get_Sink_Cap_Ext* state, from the *PE_SRC_Ready* state, due to a request to get the remote extended *Source Capabilities* from the *Device Policy Manager*.

On entry to the *PE_SRC_Get_Sink_Cap_Ext* state the *Policy Engine* **Shall** send a *Get_Sink_Cap_Extended* Message and initialize and run the *SenderResponseTimer*.

On exit from the *PE_SRC_Get_Sink_Cap_Ext* state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

The *Policy Engine* **Shall** transition back to the *PE_SRC_Ready* state (see Figure 8.132, "Source Port State Diagram") when:

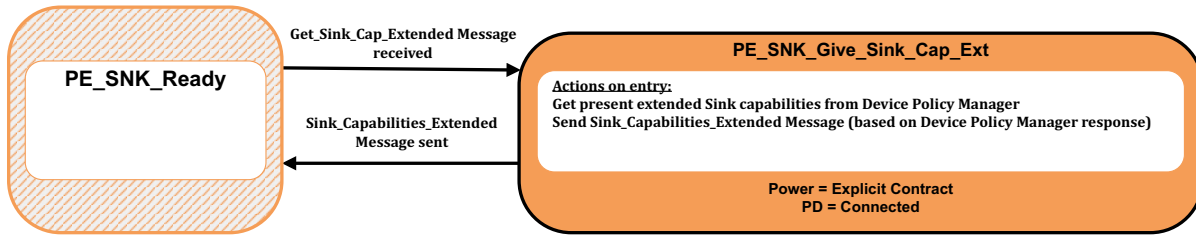
- A *Sink_Capabilities_Extended* Message is received
- Or *SenderResponseTimer* times out.

8.3.3.8.4

Sink Give Sink Capabilities Extended State Diagram

Figure 8.147, "Sink Give Sink Capabilities Extended State Diagram" shows the state diagram for a *Source* on receiving a *Get_Sink_Cap_Extended* Message. See also Section 6.5.13, "Sink Capabilities Extended Message".

Figure 8.147 Sink Give Sink Capabilities Extended State Diagram



8.3.3.8.4.1

PE_SNK_Give_Sink_Cap_Ext State

The *Policy Engine* **Shall** transition to the **PE_SNK_Give_Sink_Cap_Ext** state, from the **PE_SNK_Ready** state, when a *Get_Sink_Cap_Extended* Message is received.

On entry to the **PE_SNK_Give_Sink_Cap_Ext** state the *Policy Engine* **Shall** request the present extended *Source Capabilities* from the *Device Policy Manager* and then send a *Sink_Capabilities_Extended* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** state (see Figure 8.133, "Sink Port State Diagram") when:

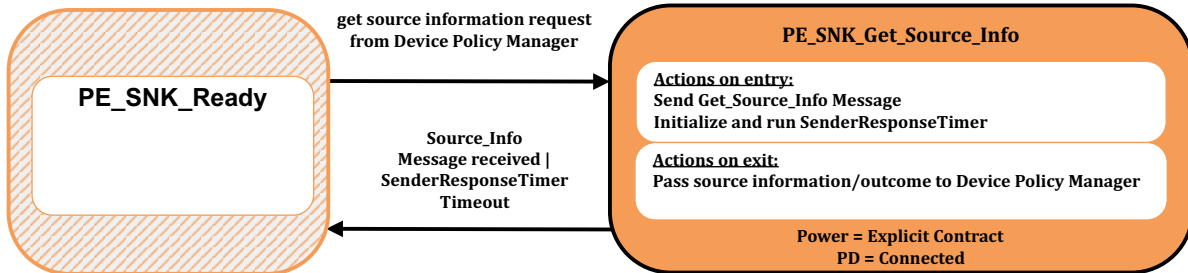
- The *Sink_Capabilities_Extended* Message has been successfully sent.

8.3.3.9 Source Information State Diagrams

8.3.3.9.1 Sink Port Get Source Information State Diagram

[Figure 8.148, "Sink Port Get Source Information State Diagram"](#) shows the state diagram for a *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner's Source* information. See also [Section 6.3.23, "Get_Source_Info Message"](#) and [Section 6.4.11, "Source_Info Message"](#).

Figure 8.148 Sink Port Get Source Information State Diagram



8.3.3.9.1.1 PE_SNK_Get_Source_Info State

The *Policy Engine* **Shall** transition to the **PE_SNK_Get_Source_Info** state, from the **PE_SNK_Ready** state, due to a request to get the remote *Source* information from the *Device Policy Manager*.

On entry to the **PE_SNK_Get_Source_Info** state the *Policy Engine* **Shall** send a **Get_Source_Info** Message and initialize and run the **SenderResponseTimer**.

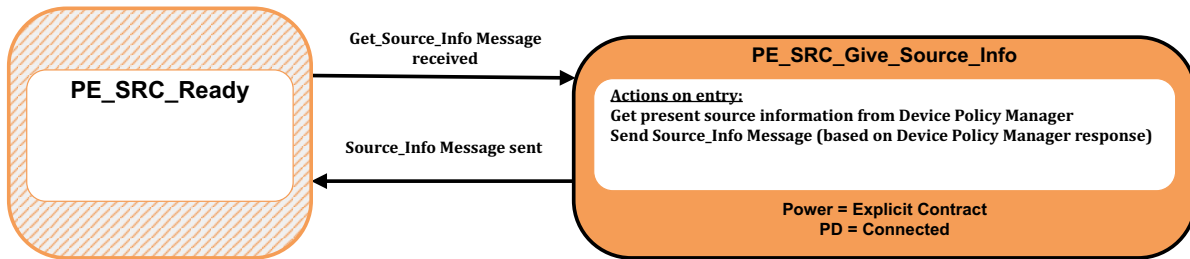
On exit from the **PE_SNK_Get_Source_Info** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (information or response timeout).

The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** state (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Source_Info** Message is received
- Or **SenderResponseTimer** times out.

Figure 8.149, "Source Give Source Information State Diagram" shows the state diagram for a *Source* on receiving a *Get_Source_Info* Message. See also [Section 6.3.23, "Get_Source_Info Message"](#) and [Section 6.4.11, "Source_Info Message"](#).

Figure 8.149 Source Give Source Information State Diagram



8.3.3.9.2.1

PE_SRC_Give_Source_Info State

The *Policy Engine* **Shall** transition to the **PE_SRC_Give_Source_Info** state, from the **PE_SRC_Ready** state, when a *Get_Source_Info* Message is received.

On entry to the **PE_SRC_Give_Source_Info** state the *Policy Engine* **Shall** request the present *Source* information from the *Device Policy Manager* and then send a *Source_Info* Message based on this information.

The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** state (see [Figure 8.132, "Source Port State Diagram"](#)) when:

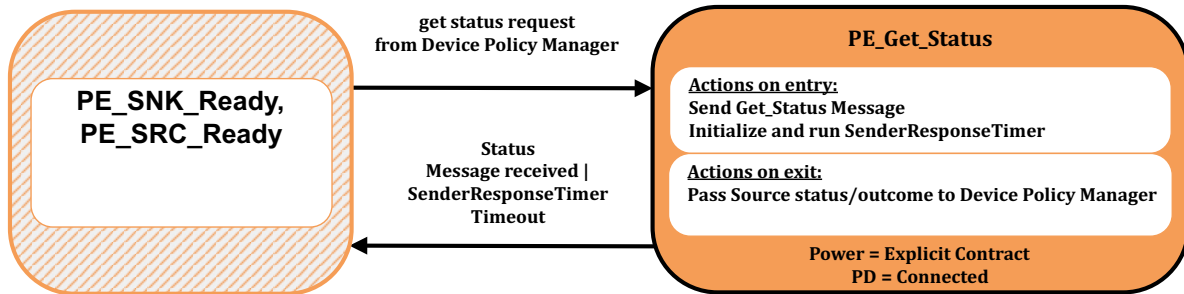
- The *Source_Info* Message has been successfully sent.

8.3.3.10 Status State Diagrams

8.3.3.10.1 Get Status State Diagram

[Figure 8.150, "Get Status State Diagram"](#) shows the state diagram for a Port on receiving a request from the *Device Policy Manager* to get the *Port Partner* or *Cable Plug*'s Status. See also [Section 6.5.2, "Status Message"](#).

Figure 8.150 Get Status State Diagram



8.3.3.10.1.1 PE_Get_Status State

The *Policy Engine* **Shall** transition to the **PE_Get_Status** state, from the **PE_SRC_Ready** or **PE_SNK_Ready** States, due to a request to get the *Port Partner* or *Cable Plug*'s status from the *Device Policy Manager*.

On entry to the **PE_Get_Status** state the *Policy Engine* **Shall** send a **Get_Status** Message and initialize and run the **SenderResponseTimer**.

On exit from the **PE_Get_Status** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (status or response timeout).

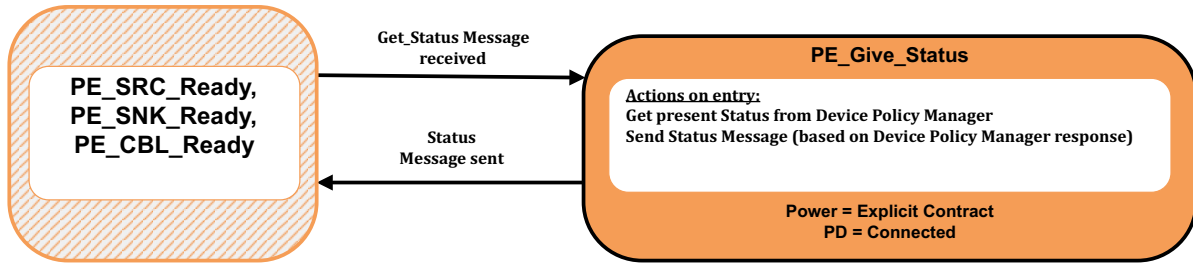
The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** or **PE_SNK_Ready** States as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) or [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Status** Message is received
- Or **SenderResponseTimer** times out.

8.3.3.10.2 Give Status State Diagram

[Figure 8.151, "Give Status State Diagram"](#) shows the state diagram for a *Source* on receiving a *Get_Status* Message. See also [Section 6.5.2, "Status Message"](#).

Figure 8.151 Give Status State Diagram



8.3.3.10.2.1 PE_Give_Status State

The *Policy Engine* **Shall** transition to the *PE_Give_Status* state, from the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* States, when a *Get_Status* Message is received.

On entry to the *PE_Give_Status* state the *Policy Engine* **Shall** request the present *Source* status from the *Device Policy Manager* and then send a *Status* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* States as appropriate (see [Figure 8.132, "Source Port State Diagram"](#), [Figure 8.133, "Sink Port State Diagram"](#) and [Figure 8.203, "Cable Ready State Diagram"](#)) when:

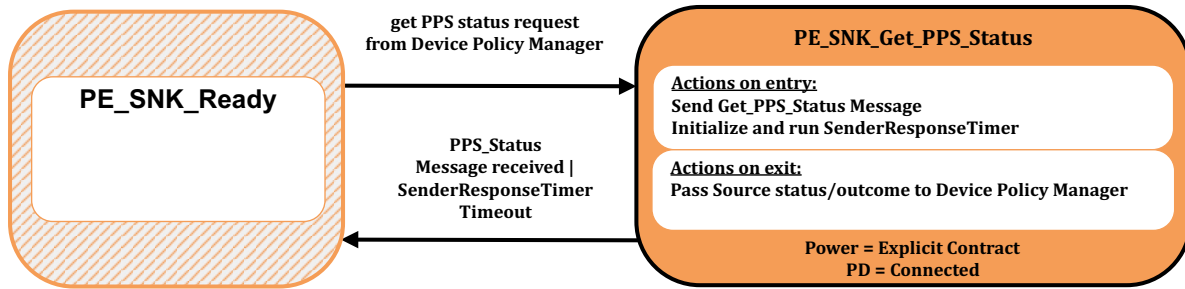
- The *Status* Message has been successfully sent.

8.3.3.10.3

Sink Port Get Source PPS Status State Diagram

[Figure 8.152, "Sink Port Get Source PPS Status State Diagram"](#) shows the state diagram for a *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner's Source* status when operating as a PPS. See also [Section 6.5.10, "PPS_Status Message"](#).

Figure 8.152 Sink Port Get Source PPS Status State Diagram



8.3.3.10.3.1

PE_SNK_Get_PPS_Status State

The *Policy Engine* **Shall** transition to the **PE_SNK_Get_PPS_Status** state, from the **PE_SNK_Ready** state, due to a request to get the remote *Source* PPS status from the *Device Policy Manager*.

On entry to the **PE_SNK_Get_PPS_Status** state the *Policy Engine* **Shall** send a **Get_PPS_Status** Message and initialize and run the **SenderResponseTimer**.

On exit from the **PE_SNK_Get_PPS_Status** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (status or response timeout).

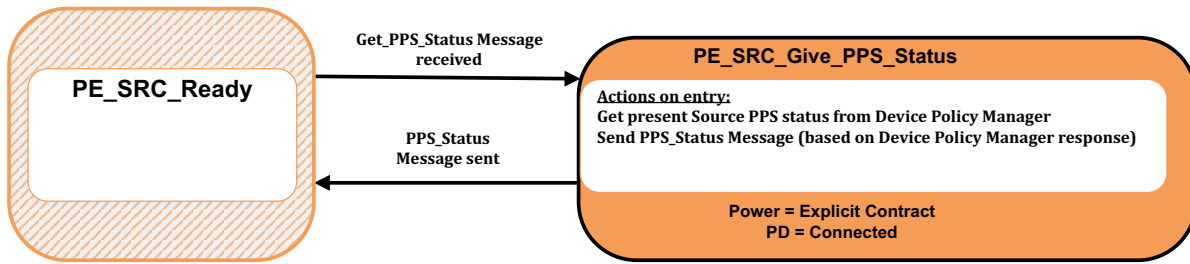
The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** state (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **PPS_Status** Message is received
- Or **SenderResponseTimer** times out.

8.3.3.10.4 Source Give Source PPS Status State Diagram

[Figure 8.153, "Source Give Source PPS Status State Diagram"](#) shows the state diagram for a *Source* on receiving a *Get_PPS_Status* Message. See also [Section 6.5.10, "PPS_Status Message"](#).

Figure 8.153 Source Give Source PPS Status State Diagram



8.3.3.10.4.1 PE_SRC_Give_PPS_Status State

The *Policy Engine* **Shall** transition to the **PE_SRC_Give_PPS_Status** state, from the **PE_SRC_Ready** state, when a *Get_PPS_Status* Message is received.

On entry to the **PE_SRC_Give_PPS_Status** state the *Policy Engine* **Shall** request the present *Source* PPS status from the *Device Policy Manager* and then send a *PPS_Status* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** state (see [Figure 8.132, "Source Port State Diagram"](#)) when:

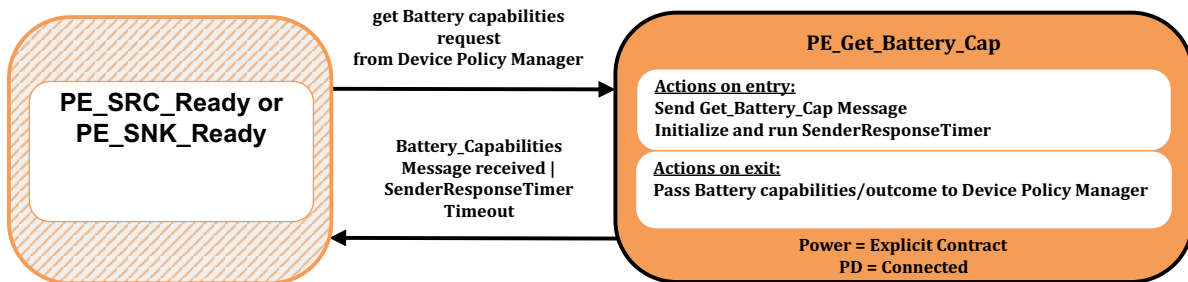
- The *PPS_Status* Message has been successfully sent.

8.3.3.11 Battery Capabilities State Diagrams

8.3.3.11.1 Get Battery Capabilities State Diagram

[Figure 8.154, "Get Battery Capabilities State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner's* *Battery* capabilities for a specified *Battery*. See also [Section 6.5.5, "Battery Capabilities Message"](#).

Figure 8.154 Get Battery Capabilities State Diagram



8.3.3.11.1.1 PE_Get_Battery_Cap State

The *Policy Engine* **Shall** transition to the **PE_Get_Battery_Cap** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to get the remote *Battery* capabilities, for a specified *Battery*, from the *Device Policy Manager*.

On entry to the **PE_Get_Battery_Cap** state the *Policy Engine* **Shall** send a **Get_Battery_Cap** Message and initialize and run the **SenderResponseTimer**.

On exit from the **PE_Get_Battery_Cap** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

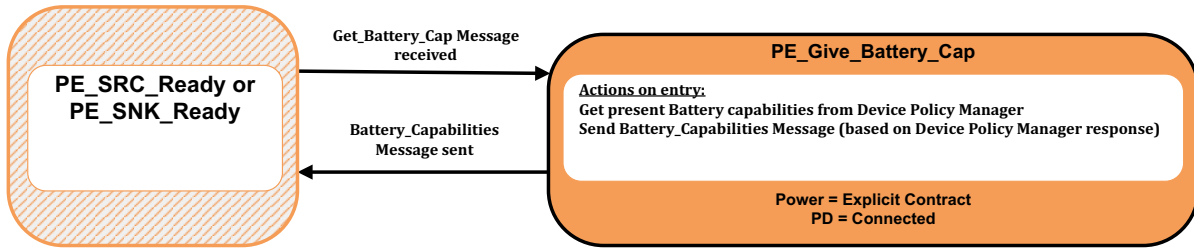
The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Battery_Capabilities** Message is received
- Or **SenderResponseTimer** times out.

8.3.3.11.2 Give Battery Capabilities State Diagram

[Figure 8.155, "Give Battery Capabilities State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a *Get_Battery_Cap* Message. See also [Section 6.5.5, "Battery Capabilities Message"](#).

Figure 8.155 Give Battery Capabilities State Diagram



8.3.3.11.2.1 PE_Give_Battery_Cap State

The *Policy Engine* **Shall** transition to the **PE_Give_Battery_Cap** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, when a *Get_Battery_Cap* Message is received.

On entry to the **PE_Give_Battery_Cap** state the *Policy Engine* **Shall** request the present *Battery* capabilities, for the requested *Battery*, from the *Device Policy Manager* and then send a *Battery_Capabilities* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

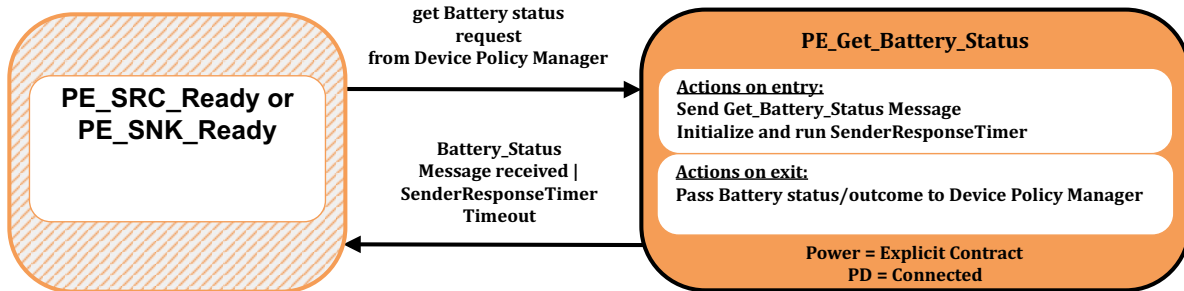
- The *Battery_Capabilities* Message has been successfully sent.

8.3.3.12 Battery Status State Diagrams

8.3.3.12.1 Get Battery Status State Diagram

[Figure 8.156, "Get Battery Status State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner's* *Battery* status for a specified *Battery*. See also [Section 6.5.4, "Get_Battery_Status Message"](#).

Figure 8.156 Get Battery Status State Diagram



8.3.3.12.1.1 PE_Get_Battery_Status State

The *Policy Engine* **Shall** transition to the **PE_Get_Battery_Status** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to get the remote *Battery* status, for a specified *Battery*, from the *Device Policy Manager*.

On entry to the **PE_Get_Battery_Status** state the *Policy Engine* **Shall** send a **Get_Battery_Status** Message and initialize and run the **SenderResponseTimer**.

On exit from the **PE_Get_Battery_Status** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (status or response timeout).

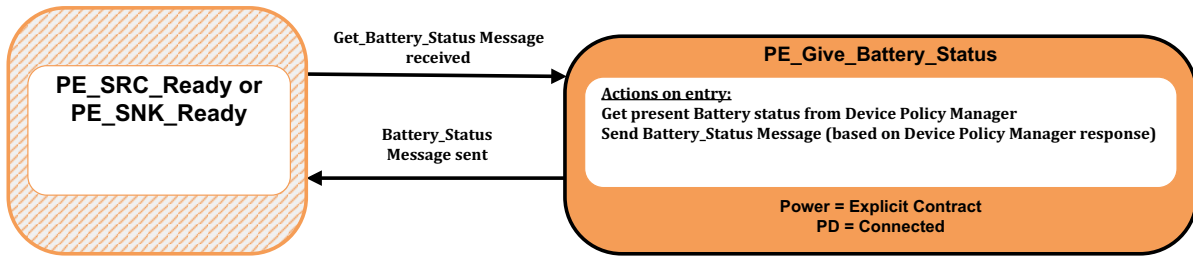
The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Battery_Status** Message is received
- Or **SenderResponseTimer** times out.

8.3.3.12.2 Give Battery Status State Diagram

[Figure 8.157, "Give Battery Status State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a *Get_Battery_Status* Message. See also [Section 6.5.4, "Get_Battery_Status Message"](#).

Figure 8.157 Give Battery Status State Diagram



8.3.3.12.2.1 PE_Give_Battery_Status State

The *Policy Engine* **Shall** transition to the **PE_Give_Battery_Status** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, when a *Get_Battery_Status* Message is received.

On entry to the **PE_Give_Battery_Status** state the *Policy Engine* **Shall** request the present *Battery* status, for the requested *Battery*, from the *Device Policy Manager* and then send a *Battery_Status* Message based on this status.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

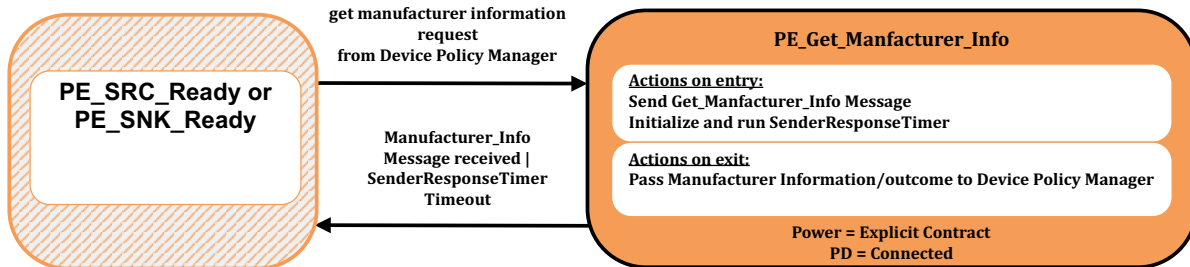
- The *Battery_Status* Message has been successfully sent.

8.3.3.13 Manufacturer Information State Diagrams

8.3.3.13.1 Get Manufacturer Information State Diagram

[Figure 8.158, "Get Manufacturer Information State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner* or *Cable Plug*'s Manufacturer Information. See also [Section 6.5.6, "Get Manufacturer Info Message"](#).

Figure 8.158 Get Manufacturer Information State Diagram



8.3.3.13.1.1 PE_Get_Manufacturer_Info State

The *Policy Engine* **Shall** transition to the **PE_Get_Manufacturer_Info** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to get the remote Manufacturer Information from the *Device Policy Manager*.

On entry to the **PE_Get_Manufacturer_Info** state the *Policy Engine* **Shall** send a **Get_Manufacturer_Info** Message and initialize and run the **SenderResponseTimer**.

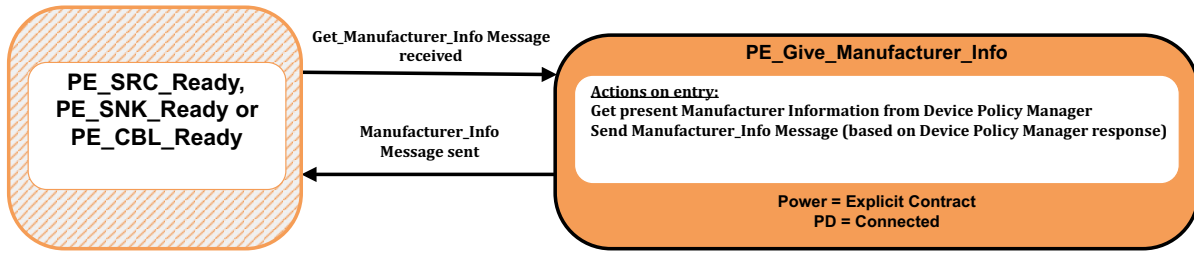
On exit from the **PE_Get_Manufacturer_Info** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (information or response timeout).

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Manufacturer_Info** Message is received
- Or **SenderResponseTimer** times out.

[Figure 8.159, "Give Manufacturer Information State Diagram"](#) shows the state diagram for a *Source*, *Sink* or *Cable Plug* on receiving a *Get_Manufacturer_Info* Message. See also [Section 6.5.6, "Get_Manufacturer_Info Message"](#).

Figure 8.159 Give Manufacturer Information State Diagram



8.3.3.13.2.1

PE_Give_Manufacturer_Info State

The *Policy Engine* **Shall** transition to the **PE_Give_Manufacturer_Info** state, from either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state, when a *Get_Manufacturer_Info* Message is received.

On entry to the **PE_Give_Manufacturer_Info** state the *Policy Engine* **Shall** request the manufacturer information from the *Device Policy Manager* and then send a *Manufacturer_Info* Message based on this status.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#), [Figure 8.133, "Sink Port State Diagram"](#) and [Figure 8.203, "Cable Ready State Diagram"](#)) when:

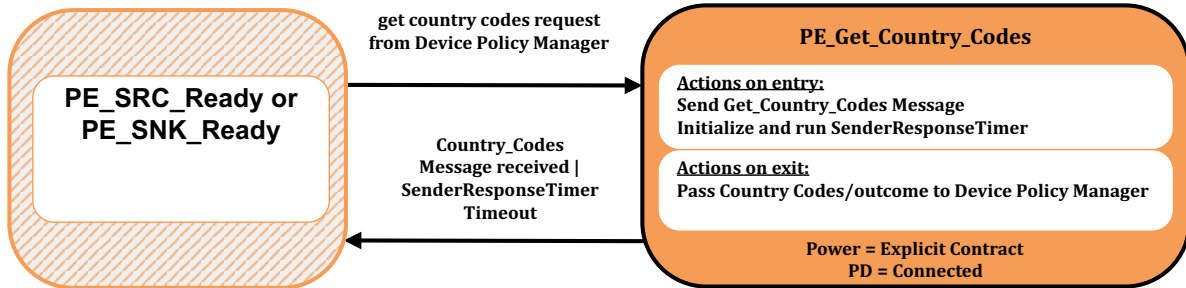
- The *Manufacturer_Info* Message has been successfully sent.

8.3.3.14 Country Codes and Information State Diagrams

8.3.3.14.1 Get Country Codes State Diagram

[Figure 8.160, "Get Country Codes State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner* or *Cable Plug*'s Country Codes. See also [Section 6.5.11, "Country Codes Message"](#).

Figure 8.160 Get Country Codes State Diagram



8.3.3.14.1.1 PE_Get_Country_Codes State

The *Policy Engine* **Shall** transition to the **PE_Get_Country_Codes** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to get the remote Country Codes from the *Device Policy Manager*.

On entry to the **PE_Get_Country_Codes** state the *Policy Engine* **Shall** send a **Get_Country_Codes** Message and initialize and run the **SenderResponseTimer**.

On exit from the **PE_Get_Country_Codes** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (Country Codes or response timeout).

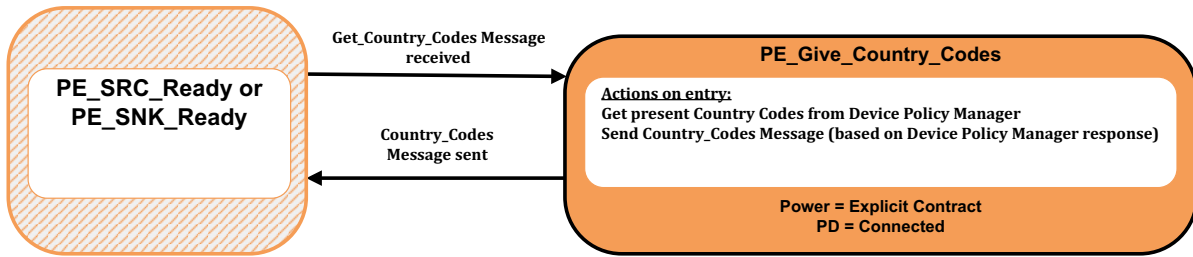
The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Country_Codes** Message is received
- Or **SenderResponseTimer** times out.

8.3.3.14.2 Give Country Codes State Diagram

[Figure 8.161, "Give Country Codes State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a *Get_Country_Codes* Message. See also [Section 6.5.11, "Country Codes Message"](#).

Figure 8.161 Give Country Codes State Diagram



8.3.3.14.2.1 PE_Give_Country_Codes State

The *Policy Engine* **Shall** transition to the **PE_Give_Country_Codes** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** State, when a *Get_Country_Codes* Message is received.

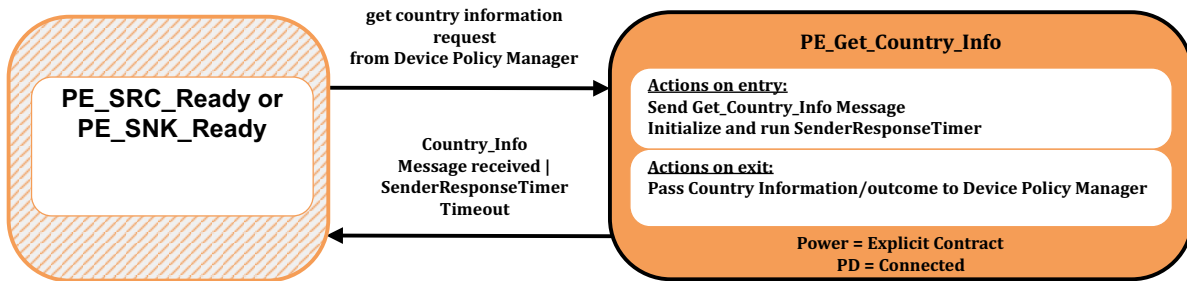
On entry to the **PE_Give_Country_Codes** state the *Policy Engine* **Shall** request the country codes from the *Device Policy Manager* and then send a *Country_Codes* Message containing these codes.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** State as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- The *Country_Codes* Message has been successfully sent.

Figure 8.162, "Get Country Information State Diagram" shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner* or *Cable Plug*'s Country Information. See also [Section 6.5.12, "Country_Info Message"](#).

Figure 8.162 Get Country Information State Diagram



8.3.3.14.3.1

PE_Get_Country_Info State

The *Policy Engine* **Shall** transition to the **PE_Get_Country_Info** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to get the remote Manufacturer Information from the *Device Policy Manager*.

On entry to the **PE_Get_Country_Info** state the *Policy Engine* **Shall** send a **Get_Manufacturer_Info** Message and initialize and run the **SenderResponseTimer**.

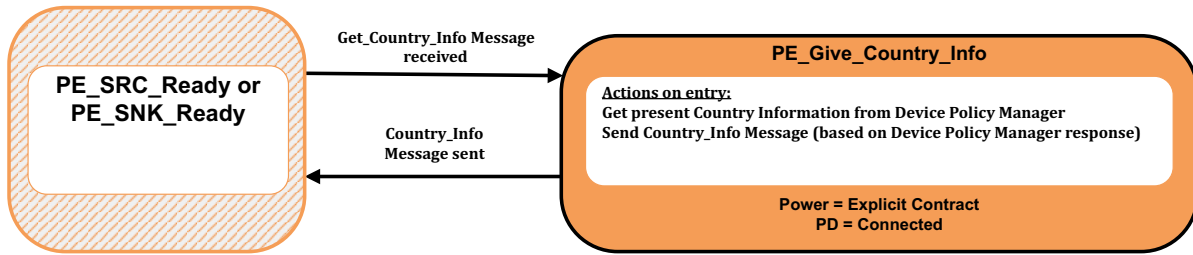
On exit from the **PE_Get_Country_Info** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (country information or response timeout).

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Country_Info** Message is received
- Or **SenderResponseTimer** times out.

[Figure 8.163, "Give Country Information State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a *Get_Country_Info* Message. See also [Section 6.5.12, "Country_Info Message"](#).

Figure 8.163 Give Country Information State Diagram



8.3.3.14.4.1

PE_Give_Country_Info State

The *Policy Engine* **Shall** transition to the *PE_Give_Country_Info* state, from either the *PE_SRC_Ready* or *PE_SNK_Ready* State, when a *Get_Country_Info* Message is received.

On entry to the *PE_Give_Country_Info* state the *Policy Engine* **Shall** request the country information from the *Device Policy Manager* and then send a *Country_Info* Message containing this country information.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready* or *PE_SNK_Ready* State as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

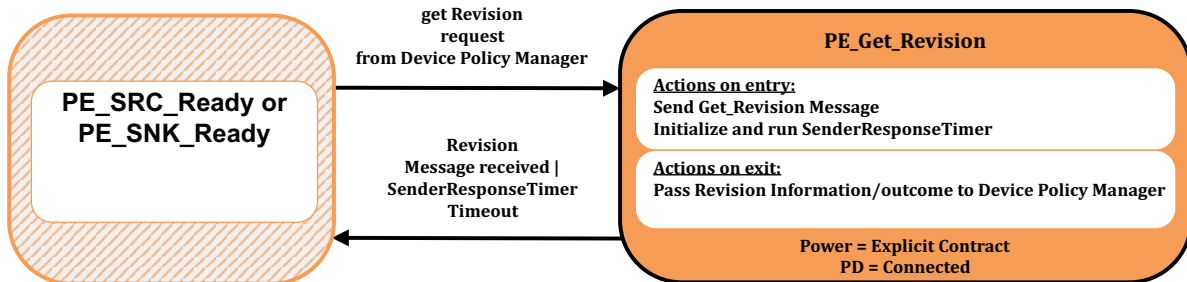
- The *Country_Info* Message has been successfully sent.

8.3.3.15 Revision State Diagrams

8.3.3.15.1 Get Revision State Diagram

[Figure 8.164, "Get Revision State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to get the *Port Partner* or *Cable Plug*'s Revision Information. See also [Section 6.3.24, "Get Revision Message"](#) and [Section 6.4.12, "Revision Message"](#).

Figure 8.164 Get Revision State Diagram



8.3.3.15.1.1 PE_Get_Revision State

The *Policy Engine* **Shall** transition to the **PE_Get_Revision** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to get the remote Revision Information from the *Device Policy Manager*.

On entry to the **PE_Get_Revision** state the *Policy Engine* **Shall** send a **Get_Revision** Message and initialize and run the **SenderResponseTimer**.

On exit from the **PE_Get_Revision** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (Revision information or response timeout).

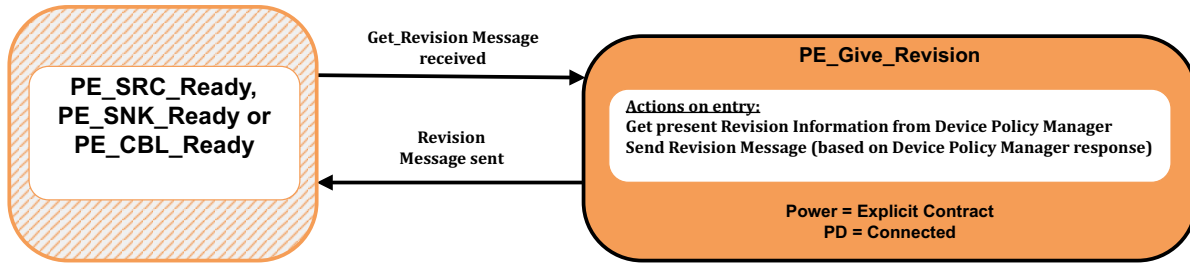
The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Revision** Message is received
- Or **SenderResponseTimer** times out.

8.3.3.15.2 Give Revision State Diagram

[Figure 8.165, "Give Revision State Diagram"](#) shows the state diagram for a *Source*, *Sink* or *Cable Plug* on receiving a *Get_Revision* Message. See also [Section 6.3.24, "Get_Revision Message"](#) and [Section 6.4.12, "Revision Message"](#).

Figure 8.165 Give Revision State Diagram



8.3.3.15.2.1 PE_Give_Revision State

The *Policy Engine* **Shall** transition to the *PE_Give_Revision* state, from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state, when a *Get_Revision* Message is received.

On entry to the *PE_Give_Revision* state the *Policy Engine* **Shall** request the Revision information from the *Device Policy Manager* and then send a *Revision* Message based on this information.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#), [Figure 8.133, "Sink Port State Diagram"](#) and [Figure 8.203, "Cable Ready State Diagram"](#)) when:

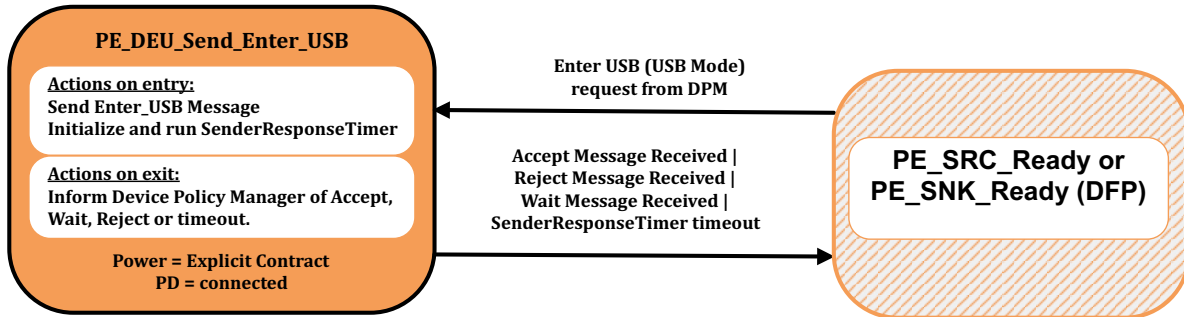
- The *Revision* Message has been successfully sent.

8.3.3.16 Enter_USB Message State Diagrams

8.3.3.16.1 DFP Enter_USB Message State Diagrams

Figure 8.166, "DFP Enter_USB Message State Diagram" shows the state diagram for an *Enter_USB* Message sent by a DFP.

Figure 8.166 DFP Enter_USB Message State Diagram



8.3.3.16.1.1 PE_DEU_Send_Enter_USB State

The *PE_DEU_Send_Enter_USB* State **Shall** be entered from the *PE_SRC_Ready* or *PE_SNK_Ready* State when requested by the *Device Policy Manager* and the Port is operating as a *DFP*.

On entry to the *PE_DEU_Send_Enter_USB* State the *Policy Engine* **Shall** request the *Protocol Layer* to send an *Enter_USB* Message and then initialize and run the *SenderResponseTimer*.

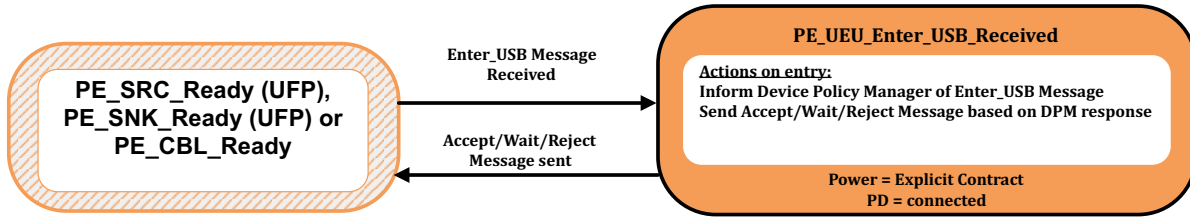
On exit from the *PE_DEU_Send_Enter_USB* state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome: *Accept* Message received, *Reject* Message received, *SenderResponseTimer* timeout.

The *Policy Engine* **Shall** transition back to the *PE_SRC_Ready* or *PE_SNK_Ready* State depending on the Ports *Power Role* when:

- An *Accept* Message has been received or
- A *Wait* Message has been received or
- A *Reject* Message has been received
- There is a *SenderResponseTimer* timeout.

[Figure 8.167, "UFP Enter_USB Message State Diagram"](#) shows the state diagram for an **Enter_USB** Message received by a *UFP* or *Cable Plug*.

Figure 8.167 UFP Enter_USB Message State Diagram



8.3.3.16.2.1

PE_UEU_Enter_USB_Received State

The **PE_UEU_Enter_USB_Received** state **Shall** be entered from the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#), [Figure 8.133, "Sink Port State Diagram"](#) and [Figure 8.203, "Cable Ready State Diagram"](#)) when an **Enter_USB** Message is received and the Port is operating as a *UFP* or is a *Cable Plug*.

On entry to the **PE_UEU_Enter_USB_Received** state the *Policy Engine* **Shall** inform the *Device Policy Manager*. The *Device Policy Manager* responds with an indication of whether the **Enter_USB** Message is to be accepted or rejected. The *Policy Engine* **Shall** send either an **Accept** Message, a **Wait** Message or a **Reject** Message as appropriate.

The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state as appropriate when:

- Either an **Accept** Message, a **Wait** Message or a **Reject** Message has been sent.

8.3.3.17 Security State Diagrams

8.3.3.17.1 Send Security Request State Diagram

[Figure 8.168, "Send security request State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to send a security request. See also [Section 6.5.8, "Security Messages"](#).

Figure 8.168 Send security request State Diagram



8.3.3.17.1.1 PE_Send_Security_Request State

The *Policy Engine* **Shall** transition to the **PE_Send_Security_Request** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to send a security request from the *Device Policy Manager*.

On entry to the **PE_Send_Security_Request** state the *Policy Engine* **Shall** send a **Security_Request** Message.

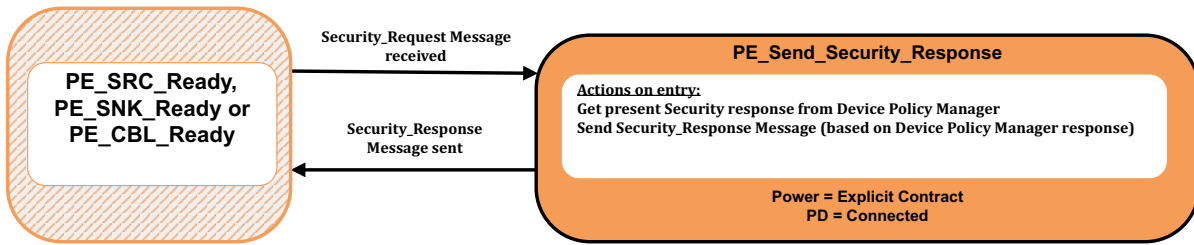
The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- The **Security_Request** Message has been sent.

8.3.3.17.2 Send Security Response State Diagram

Figure 8.169, "Send security response State Diagram" shows the state diagram for a Source, Sink or Cable Plug on receiving a *Security_Request* Message. See also Section 6.5.8, "Security Messages".

Figure 8.169 Send security response State Diagram



8.3.3.17.2.1 PE_Send_Security_Response State

The Policy Engine **Shall** transition to the *PE_Send_Security_Response* state, from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state, when a *Security_Request* Message is received.

On entry to the *PE_Send_Security_Response* state the Policy Engine **Shall** request the appropriate response from the Device Policy Manager and then send a *Security_Response* Message based on this status.

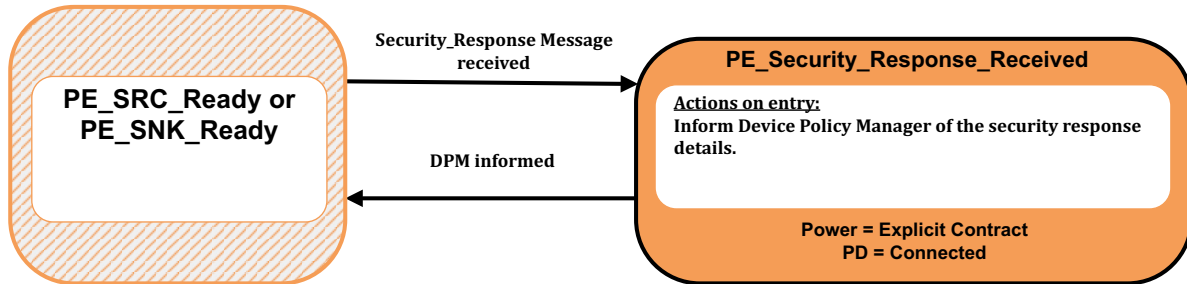
The Policy Engine **Shall** transition back to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state as appropriate (see Figure 8.132, "Source Port State Diagram", Figure 8.133, "Sink Port State Diagram" and Figure 8.203, "Cable Ready State Diagram") when:

- The *Security_Response* Message has been successfully sent.

8.3.3.17.3 Security Response Received State Diagram

[Figure 8.170, "Security response received State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a *Security_Response* Message. See also [Section 6.5.8, "Security Messages"](#).

Figure 8.170 Security response received State Diagram



8.3.3.17.3.1 PE_Security_Response_Received State

The *Policy Engine* **Shall** transition to the *PE_Security_Response_Received* state, from either the *PE_SRC_Ready* or *PE_SNK_Ready* when a *Security_Response* Message is received.

On entry to the *PE_Security_Response_Received* state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the details of the security response.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready* or *PE_SNK_Ready* state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#), [Figure 8.133, "Sink Port State Diagram"](#) and [Figure 8.203, "Cable Ready State Diagram"](#)) when:

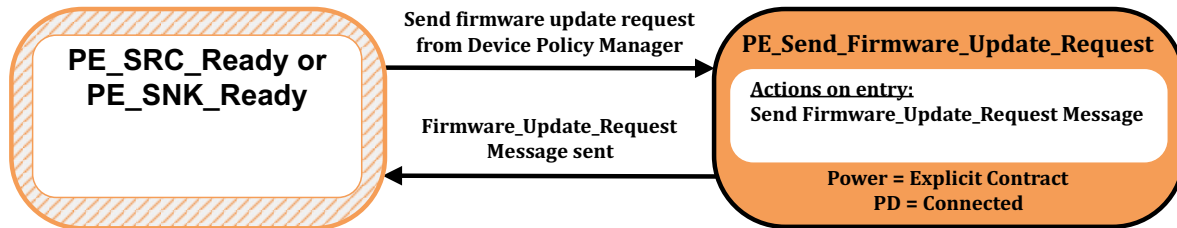
- The *Device Policy Manager* has been informed.

8.3.3.18 Firmware Update State Diagrams

8.3.3.18.1 Send Firmware Update Request State Diagram

[Figure 8.171, "Send firmware update request State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a request from the *Device Policy Manager* to send a firmware update request. See also [Section 6.5.9, "Firmware Update Messages"](#).

Figure 8.171 Send firmware update request State Diagram



8.3.3.18.1.1 PE_Send_Firmware_Update_Request State

The *Policy Engine* **Shall** transition to the **PE_Send_Firmware_Update_Request** state, from either the **PE_SRC_Ready** or **PE_SNK_Ready** state, due to a request to send a firmware update request from the *Device Policy Manager*.

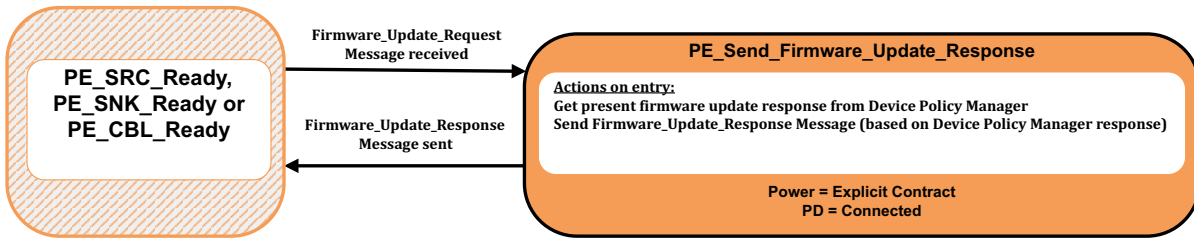
On entry to the **PE_Send_Firmware_Update_Request** state the *Policy Engine* **Shall** send a **Firmware_Update_Request** Message.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready** or **PE_SNK_Ready** state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#) and [Figure 8.133, "Sink Port State Diagram"](#)) when:

- The **Firmware_Update_Request** Message has been sent.

Figure 8.172, "Send firmware update response State Diagram" shows the state diagram for a Source, Sink or Cable Plug on receiving a *Firmware_Update_Request* Message. See also Section 6.5.9, "Firmware Update Messages".

Figure 8.172 Send firmware update response State Diagram



8.3.3.18.2.1

PE_Send_Firmware_Update_Response State

The Policy Engine **Shall** transition to the *PE_Send_Firmware_Update_Response* state, from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state, when a *Firmware_Update_Request* Message is received.

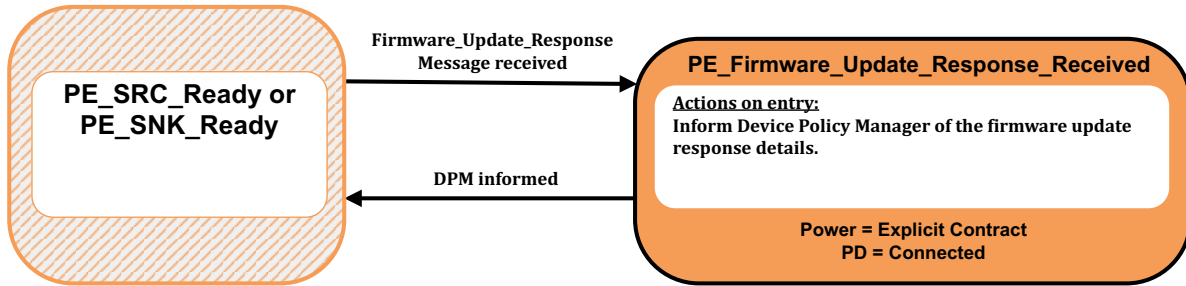
On entry to the *PE_Send_Firmware_Update_Response* state the Policy Engine **Shall** request the appropriate response from the Device Policy Manager and then send a *Firmware_Update_Response* Message based on this status.

The Policy Engine **Shall** transition back to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state as appropriate (see Figure 8.132, "Source Port State Diagram", Figure 8.133, "Sink Port State Diagram" and Figure 8.203, "Cable Ready State Diagram") when:

- The *Firmware_Update_Response* Message has been successfully sent.

[Figure 8.173, "Firmware update response received State Diagram"](#) shows the state diagram for a *Source* or *Sink* on receiving a *Firmware_Update_Response* Message. See also [Section 6.5.9, "Firmware Update Messages"](#).

Figure 8.173 Firmware update response received State Diagram



8.3.3.18.3.1

PE_Firmware_Update_Response_Received State

The *Policy Engine* **Shall** transition to the *PE_Firmware_Update_Response_Received* state, from either the *PE_SRC_Ready* or *PE_SNK_Ready* when a *Firmware_Update_Response* Message is received.

On entry to the *PE_Firmware_Update_Response_Received* state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the details of the firmware update response.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready* or *PE_SNK_Ready* state as appropriate (see [Figure 8.132, "Source Port State Diagram"](#), [Figure 8.133, "Sink Port State Diagram"](#) and [Figure 8.203, "Cable Ready State Diagram"](#)) when:

- The *Device Policy Manager* has been informed.

8.3.3.19 Dual-Role Port State Diagrams

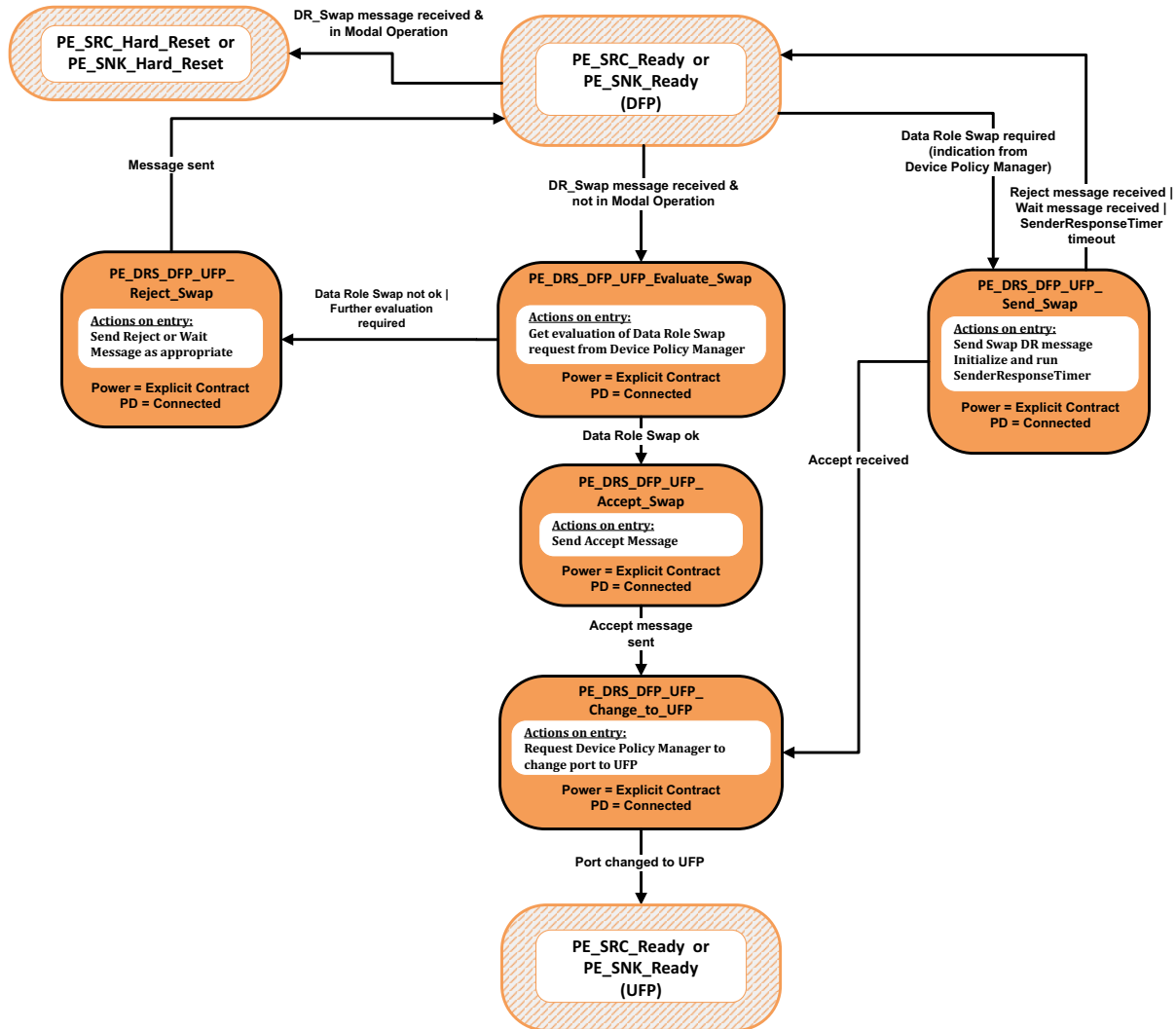
Dual-Role Ports that combine *Source* and *Sink* functionality **shall** comprise *Source* and *Sink Policy Engine* state machines. In addition they **shall** have the capability to perform a *Power Role Swap* from the **PE_SRC_Ready** or **PE_SNK_Ready** states and **shall** return to *USB Default Operation* on a *Hard Reset*.

The State Diagrams in this section **shall** apply to every [USB Type-C 2.4] *DRP*.

8.3.3.19.1 DFP to UFP Data Role Swap State Diagram

Figure 8.174, "*DFP to UFP Data Role Swap State Diagram*" shows the additional state diagram required to perform a *Data Role Swap* from *DFP* to *UFP* operation and the changes that **shall** be followed for error and *Hard Reset* handling.

Figure 8.174 DFP to UFP Data Role Swap State Diagram



8.3.3.19.1.1 PE_SRC_Ready or PE_SNK_Ready State

The *Data Role Swap* process **shall** start only from either the **PE_SRC_Ready** or **PE_SNK_Ready** state where power is stable.

The *Policy Engine* **shall** transition to the **PE_DRS_DFP_UFP_Evaluate_Swap** state when:

- A **DR_Swap** Message is received and

- There are no *Active Modes* (not in *Modal Operation*).

The *Policy Engine* **Shall** transition to either the *PE_SRC_Hard_Reset* or *PE_SNK_Hard_Reset* states when:

- A *DR_Swap* Message is received and
- There are one or more *Active Modes* (*Modal Operation*).

The *Policy Engine* **Shall** transition to the *PE_DRS_DFP_UFP_Send_Swap* state when:

- The *Device Policy Manager* indicates that a *Data Role Swap* is required.

8.3.3.19.1.2 PE_DRS_DFP_UFP_Evaluate_Swap State

On entry to the *PE_DRS_DFP_UFP_Evaluate_Swap* state the *Policy Engine* **Shall** ask the *Device Policy Manager* whether a *Data Role Swap* can be made.

The *Policy Engine* **Shall** transition to the *PE_DRS_DFP_UFP_Accept_Swap* state when:

- The *Device Policy Manager* indicates that a *Data Role Swap* is OK.

The *Policy Engine* **Shall** transition to the *PE_DRS_DFP_UFP_Reject_Swap* state when:

- The *Device Policy Manager* indicates that a *Data Role Swap* is not OK.
- Or further evaluation of the *Data Role Swap* request is needed.

8.3.3.19.1.3 PE_DRS_DFP_UFP_Accept_Swap State

On entry to the *PE_DRS_DFP_UFP_Accept_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send an *Accept* Message.

The *Policy Engine* **Shall** transition to the *PE_DRS_DFP_UFP_Change_to_UFP* state when:

- The *Accept* Message has been sent.

8.3.3.19.1.4 PE_DRS_DFP_UFP_Change_to_UFP State

On entry to the *PE_DRS_DFP_UFP_Change_to_UFP* state the *Policy Engine* **Shall** request the *Device Policy Manager* to change the Port from a *DFP* to a *UFP*.

The *Policy Engine* **Shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state when:

- The *Device Policy Manager* indicates that the Port has been changed to a *UFP*.

8.3.3.19.1.5 PE_DRS_DFP_UFP_Send_Swap State

On entry to the *PE_DRS_DFP_UFP_Send_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send a *DR_Swap* Message and **Shall** start the *SenderResponseTimer*.

On exit from the *PE_DRS_DFP_UFP_Send_Swap* state the *Policy Engine* **Shall** stop the *SenderResponseTimer*.

The *Policy Engine* **Shall** continue as a *DFP* and **Shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state when:

- A *Reject* Message is received.
- Or a *Wait* Message is received.
- Or the *SenderResponseTimer* times out.

The *Policy Engine* **Shall** transition to the *PE_DRS_DFP_UFP_Change_to_UFP* state when:

- An *Accept* Message is received.

8.3.3.19.1.6 PE_DRS_DFP_UFP_Reject_Swap State

On entry to the *PE_DRS_DFP_UFP_Reject_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send:

- A **Reject** Message if the device is unable to perform a *Data Role Swap* at this time.
- A **Wait** Message if further evaluation of the *Data Role Swap* request is required.

Note: In this case it is expected that one of the *Port Partners* will send a **DR_Swap** Message at a later time (see [Section 6.3.12.3, "Wait in response to a DR_Swap Message"](#)).

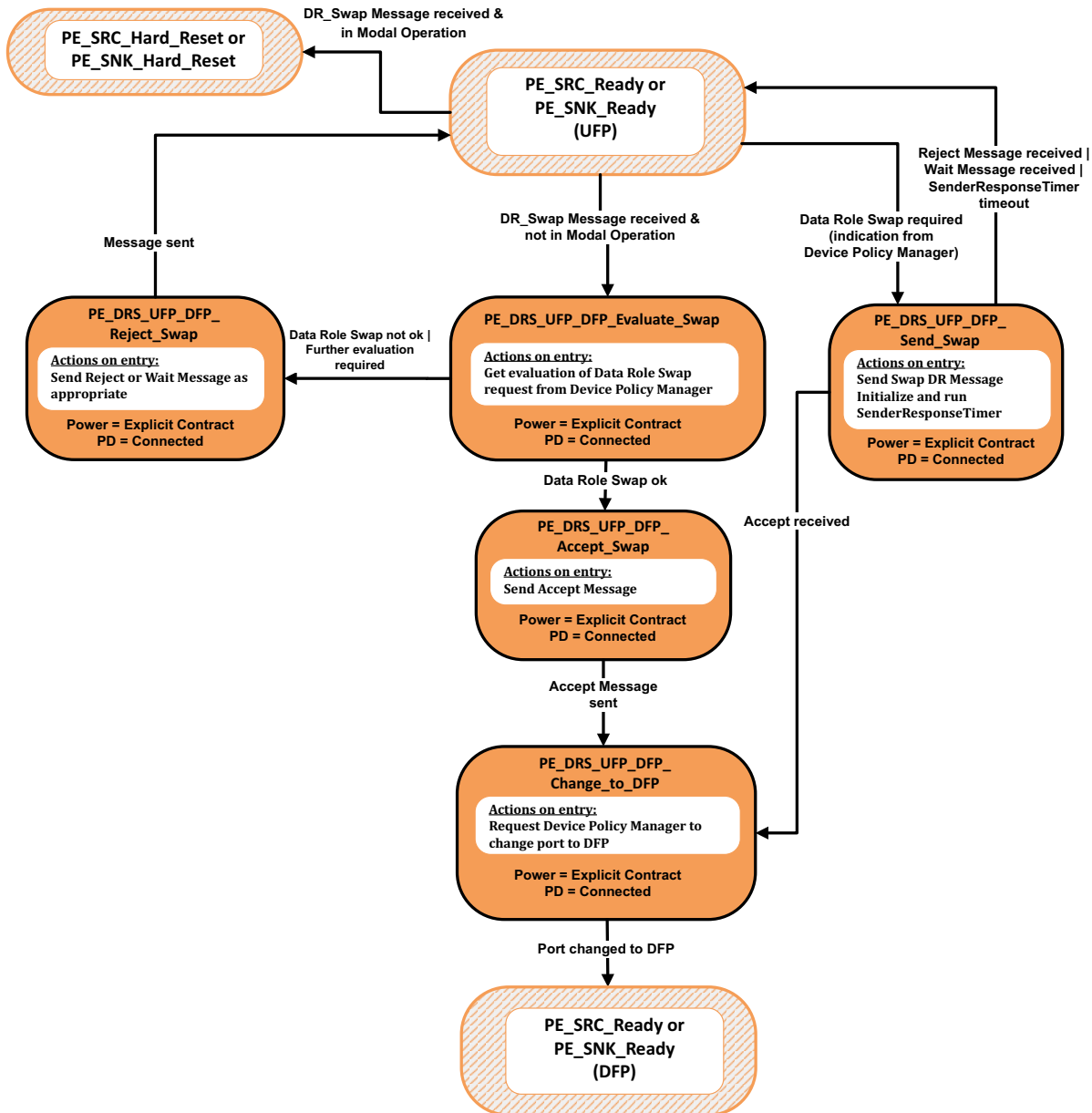
The *Policy Engine* **Shall** continue as a *DFP* and **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The **Reject** or **Wait** Message has been sent.

8.3.3.19.2 UFP to DFP Data Role Swap State Diagram

Figure 8.175, "UFP to DFP Data Role Swap State Diagram" shows the additional state diagram required to perform a *Data Role Swap* from *DRP UFP* to *DFP* operation and the changes that **Shall** be followed for error and *Hard Reset* handling.

Figure 8.175 UFP to DFP Data Role Swap State Diagram



8.3.3.19.2.1 PE_SRC_Ready or PE_SNK_Ready State

The *Data Role Swap* process **Shall** start only from the either the *PE_SRC_Ready* or *PE_SNK_Ready* state where power is stable.

The *Policy Engine* **Shall** transition to the *PE_DRP_UFP_DFP_Evaluate_Swap* state when:

- A *DR_Swap* Message is received and
- There are no *Active Modes* (not in *Modal Operation*).

The *Policy Engine* **Shall** transition to either the *PE_SRC_Hard_Reset* or *PE_SNK_Hard_Reset* states when:

- A *DR_Swap* Message is received and
- There are one or more *Active Modes* (Modal Operation).

The *Policy Engine* **Shall** transition to the *PE_DRS_UFP_DFP_Send_Swap* state when:

- The *Device Policy Manager* indicates that a *Data Role Swap* is required.

8.3.3.19.2.2 PE_DRS_UFP_DFP_Evaluate_Swap State

On entry to the *PE_DRS_UFP_DFP_Evaluate_Swap* state the *Policy Engine* **Shall** ask the *Device Policy Manager* whether a *Data Role Swap* can be made.

The *Policy Engine* **Shall** transition to the *PE_DRS_UFP_DFP_Accept_Swap* state when:

- The *Device Policy Manager* indicates that a *Data Role Swap* is OK.

The *Policy Engine* **Shall** transition to the *PE_DRS_UFP_DFP_Reject_Swap* state when:

- The *Device Policy Manager* indicates that a *Data Role Swap* is not OK.
- Or further evaluation of the *Data Role Swap* request is needed.

8.3.3.19.2.3 PE_DRS_UFP_DFP_Accept_Swap State

On entry to the *PE_DRS_UFP_DFP_Accept_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send an *Accept* Message.

The *Policy Engine* **Shall** transition to the *PE_DRS_UFP_DFP_Change_to_DFP* state when:

- The *Accept* Message has been sent.

8.3.3.19.2.4 PE_DRS_UFP_DFP_Change_to_DFP State

On entry to the *PE_DRS_UFP_DFP_Change_to_DFP* state the *Policy Engine* **Shall** request the *Device Policy Manager* to change the Port from a *UFP* to a *DFP*.

The *Policy Engine* **Shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state when:

- The *Device Policy Manager* indicates that the Port has been changed to a *DFP*.

8.3.3.19.2.5 PE_DRS_UFP_DFP_Send_Swap State

On entry to the *PE_DRS_UFP_DFP_Send_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send a *DR_Swap* Message and **Shall** start the *SenderResponseTimer*.

On exit from the *PE_DRS_UFP_DFP_Send_Swap* state the *Policy Engine* **Shall** stop the *SenderResponseTimer*.

The *Policy Engine* **Shall** continue as a *UFP* and **Shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state when:

- A *Reject* Message is received.
- Or a *Wait* Message is received.
- Or the *SenderResponseTimer* times out.

The *Policy Engine* **Shall** transition to the *PE_DRS_UFP_DFP_Change_to_DFP* state when:

- An *Accept* Message is received.

8.3.3.19.2.6 PE_DRS_UFP_DFP_Reject_Swap State

On entry to the *PE_DRS_UFP_DFP_Reject_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send:

- A *Reject* Message if the device is unable to perform a *Data Role Swap* at this time.

- A **Wait** Message if further evaluation of the *Data Role Swap* request is required.

Note: In this case it is expected that one of the *Port Partners* will send a **DR_Swap** Message at a later time (see [Section 6.3.12.3, "Wait in response to a DR Swap Message"](#)).

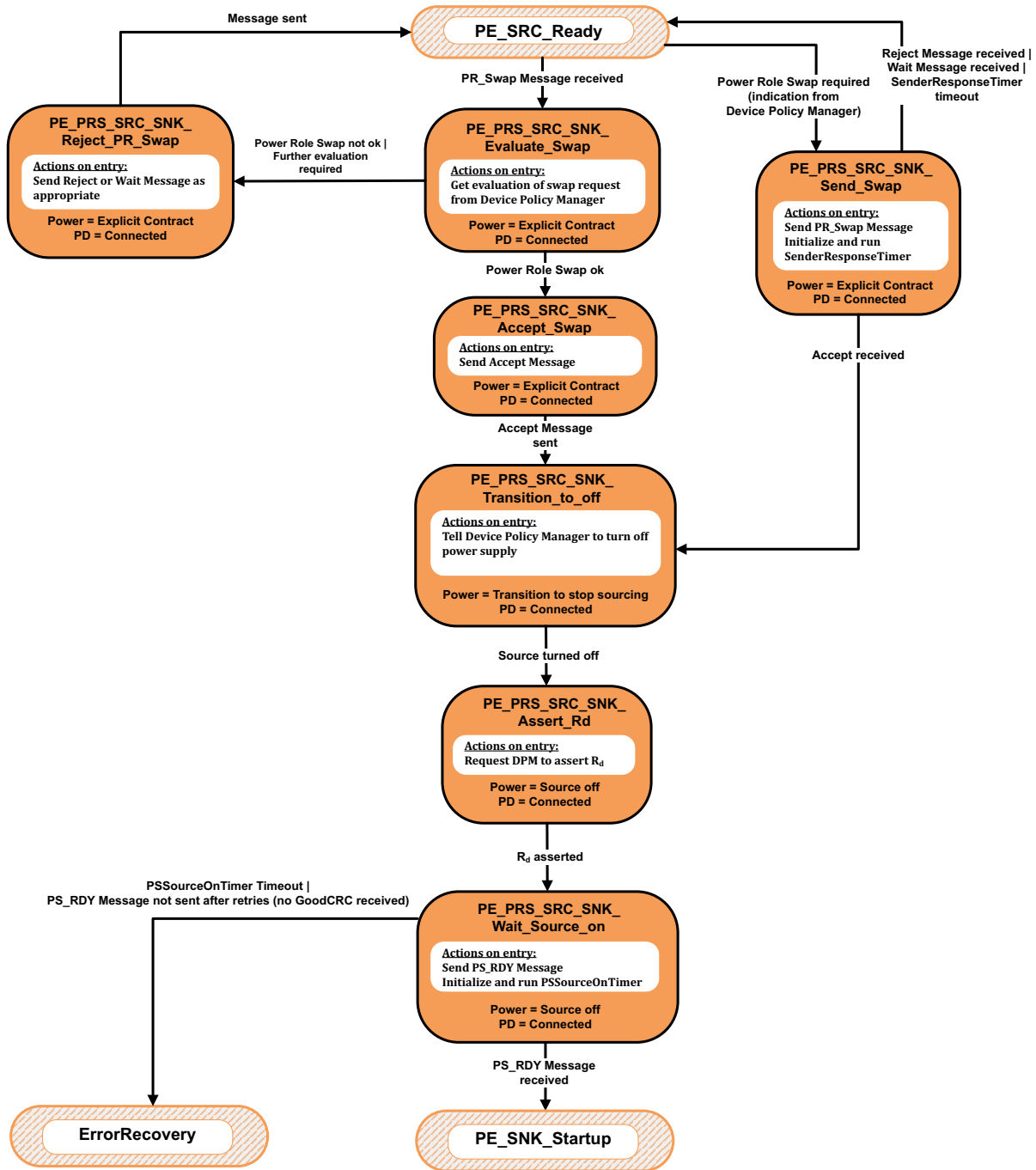
The *Policy Engine* **Shall** continue as a *UFP* and **Shall** transition to the either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The **Reject** or **Wait** Message has been sent.

Dual-Role Ports that combine *Source* and *Sink* functionality **shall** comprise *Source* and *Sink* Policy Engine state machines. In addition, they **shall** have the capability to do a *Power Role Swap* from the **PE_SRC_Ready** state and **shall** return to *USB Default Operation* on a *Hard Reset*.

Figure 8.176, "[Dual-Role Port in Source to Sink Power Role Swap State Diagram](#)" shows the additional state diagram required to perform a *Power Role Swap* from *Source* to *Sink* Power Roles and the changes that **shall** be followed for error handling.

Figure 8.176 Dual-Role Port in Source to Sink Power Role Swap State Diagram



8.3.3.19.3.1 PE_SRC_Ready State

The *Power Role Swap* process **shall** start only from the *PE_SRC_Ready* state where power is stable.

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Evaluate_Swap* state when:

- A *PR_Swap* Message is received.

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Send_Swap* state when:

- The *Device Policy Manager* indicates that a *Power Role Swap* is required.

8.3.3.19.3.2 PE_PRS_SRC_SNK_Evaluate_Swap State

On entry to the *PE_PRS_SRC_SNK_Evaluate_Swap* state the *Policy Engine* **shall** ask the *Device Policy Manager* whether a *Power Role Swap* can be made.

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Accept_Swap* state when:

- The *Device Policy Manager* indicates that a *Power Role Swap* is OK.

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Reject_Swap* state when:

- The *Device Policy Manager* indicates that a *Power Role Swap* is not OK.
- Or further evaluation of the *Power Role Swap* request is needed.

8.3.3.19.3.3 PE_PRS_SRC_SNK_Accept_Swap State

On entry to the *PE_PRS_SRC_SNK_Accept_Swap* state the *Policy Engine* **shall** request the *Protocol Layer* to send an *Accept* Message.

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Transition_to_off* state when:

- The *Accept* Message has been sent.

8.3.3.19.3.4 PE_PRS_SRC_SNK_Transition_to_off State

On entry to the *PE_PRS_SRC_SNK_Transition_to_off* state the *Policy Engine* **shall** request the *Device Policy Manager* to turn off the *Source*.

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Assert_Rd* state when:

- The *Device Policy Manager* indicates that the *Source* has been turned off.

8.3.3.19.3.5 PE_PRS_SRC_SNK_Assert_Rd State

On entry to the *PE_PRS_SRC_SNK_Assert_Rd* state the *Policy Engine* **shall** request the *Device Policy Manager* to change the resistor asserted on the *CC* wire from R_p to R_d .

The *Policy Engine* **shall** transition to the *PE_PRS_SRC_SNK_Wait_Source_on* state when:

- The *Device Policy Manager* indicates that R_d is asserted.

8.3.3.19.3.6 PE_PRS_SRC_SNK_Wait_Source_on State

On entry to the *PE_PRS_SRC_SNK_Wait_Source_on* state the *Policy Engine* **shall** request the *Protocol Layer* to send a *PS_RDY* Message and **shall** start the *PSSourceOnTimer*.

On exit from the *Source* off state the *Policy Engine* **shall** stop the *PSSourceOnTimer*.

The *Policy Engine* **shall** transition to the *PE_SNK_Startup* when:

- A *PS_RDY* Message is received indicating that the remote *Source* is now supplying power.

The *Policy Engine* **shall** transition to the *ErrorRecovery* state when:

- The *PSSourceOnTimer* times out or
- The *PS_RDY* Message is not sent after retries (a *GoodCRC* Message has not been received).

Note: A Soft Reset **Shall Not** be initiated in this case.

8.3.3.19.3.7 PE_PRS_SRC_SNK_Send_Swap State

On entry to the *PE_PRS_SRC_SNK_Send_Swap* state the Policy Engine **Shall** request the Protocol Layer to send a *PR_Swap* Message and **Shall** start the *SenderResponseTimer*.

On exit from the *PE_PRS_SRC_SNK_Send_Swap* state the Policy Engine **Shall** stop the *SenderResponseTimer*.

The Policy Engine **Shall** transition to the *PE_SRC_Ready* state when:

- A *Reject* Message is received.
- Or a *Wait* Message is received.
- Or the *SenderResponseTimer* times out.

The Policy Engine **Shall** transition to the *PE_PRS_SRC_SNK_Transition_to_off* state when:

- An *Accept* Message is received.

8.3.3.19.3.8 PE_PRS_SRC_SNK_Reject_Swap State

On entry to the *PE_PRS_SRC_SNK_Reject_Swap* state the Policy Engine **Shall** request the Protocol Layer to send:

- A *Reject* Message if the device is unable to perform a Power Role Swap at this time.
- A *Wait* Message if further evaluation of the Power Role Swap request is required.

Note: In this case it is expected that one of the Port Partners will send a *PR_Swap* Message at a later time (see [Section 6.3.12.2, "Wait in response to a PR_Swap Message"](#)).

The Policy Engine **Shall** transition to the *PE_SRC_Ready* when:

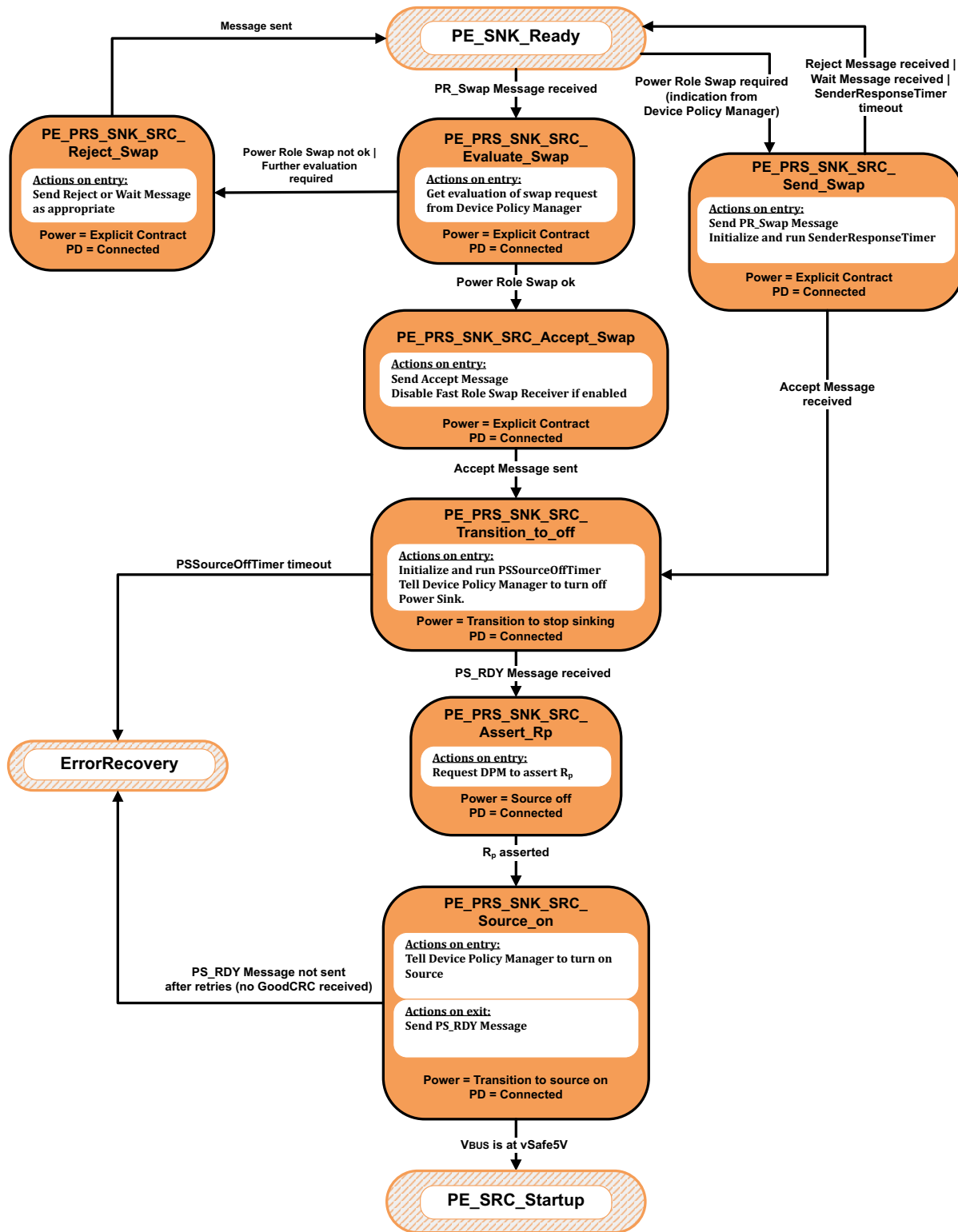
- The *Reject* or *Wait* Message has been sent.

8.3.3.19.4 Policy Engine in Sink to Source Power Role Swap State Diagram

Dual-Role Ports that combine *Sink* and *Source* functionality **shall** comprise *Sink* and *Source Policy Engine* state machines. In addition, they **shall** have the capability to do a *Power Role Swap* from the **PE_SNK_Ready** state and **shall** return to *USB Default Operation* on a *Hard Reset*.

[Figure 8.177, "Dual-role Port in Sink to Source Power Role Swap State Diagram"](#) shows the additional state diagram required to perform a *Power Role Swap* from *Sink* to *Source Power Roles* and the changes that **shall** be followed for error handling.

Figure 8.177 Dual-role Port in Sink to Source Power Role Swap State Diagram



8.3.3.19.4.1 PE_SNK_Ready State

The *Power Role Swap* process **shall** start only from the *PE_SNK_Ready* state where power is stable.

The *Policy Engine* **shall** transition to the *PE_PRS_SNK_SRC_Evaluate_Swap* state when:

- A **PR_Swap** Message is received.

The Policy Engine **Shall** transition to the **PE_PRS_SNK_SRC_Send_Swap** state when:

- The Device Policy Manager indicates that a Power Role Swap is required.

8.3.3.19.4.2 PE_PRS_SNK_SRC_Evaluate_Swap State

On entry to the **PE_PRS_SNK_SRC_Send_Swap** state the Policy Engine **Shall** ask the Device Policy Manager whether a Power Role Swap can be made.

The Policy Engine **Shall** transition to the **PE_PRS_SNK_SRC_Accept_Swap** state when:

- The Device Policy Manager indicates that a Power Role Swap is OK.

The Policy Engine **Shall** transition to the **PE_PRS_SNK_SRC_Reject_Swap** state when:

- The Device Policy Manager indicates that a Power Role Swap is not OK.

8.3.3.19.4.3 PE_PRS_SNK_SRC_Accept_Swap State

On entry to the **PE_PRS_SNK_SRC_Accept_Swap** state the Policy Engine **Shall** request the Protocol Layer to send an **Accept** Message and **Shall** disable the Fast Role Swap receiver if this is enabled.

The Policy Engine **Shall** transition to the **PE_PRS_SNK_SRC_Transition_to_off** state when:

- The **Accept** Message has been sent.

8.3.3.19.4.4 PE_PRS_SNK_SRC_Transition_to_off State

On entry to the **PE_PRS_SNK_SRC_Transition_to_off** state the Policy Engine **Shall** initialize and run the **PSSourceOffTimer** and then request the Device Policy Manager to turn off the Sink.

The Policy Engine **Shall** transition to the **ErrorRecovery** state when:

- The **PSSourceOffTimer** times out.

The Policy Engine **Shall** transition to the **PE_PRS_SNK_SRC_Assert_Rp** state when:

- A **PS_RDY** Message is received.

8.3.3.19.4.5 PE_PRS_SNK_SRC_Assert_Rp State

On entry to the **PE_PRS_SNK_SRC_Assert_Rp** state the Policy Engine **Shall** request the Device Policy Manager to change the resistor asserted on the CC wire from R_d to R_p .

The Policy Engine **Shall** transition to the **PE_PRS_SNK_SRC_Source_on** state when:

- The Device Policy Manager indicates that R_d is asserted.

8.3.3.19.4.6 PE_PRS_SNK_SRC_Source_on State

On entry to the **PE_PRS_SNK_SRC_Source_on** state the Policy Engine **Shall** request the Device Policy Manager to turn on the Source.

On exit from the **PE_PRS_SNK_SRC_Source_on** state the Policy Engine **Shall** send a **PS_RDY** Message.

The Policy Engine **Shall** transition to the **PE_SRC_Startup** state when:

- The Source Port VBUS is at **vSafe5V**.

The Policy Engine **Shall** transition to the **ErrorRecovery** state when:

- The **PS_RDY** Message is not sent after retries (a **GoodCRC** Message has not been received). A Soft Reset **Shall Not** be initiated in this case.

8.3.3.19.4.7 PE_PRS_SNK_SRC_Send_Swap State

On entry to the **PE_PRS_SNK_SRC_Send_Swap** state the *Policy Engine* **Shall** request the *Protocol Layer* to send a **PR_Swap** Message and **Shall** initialize and run the **SenderResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_SNK_Ready** state when:

- A **Reject** Message is received.
- Or a **Wait** Message is received.
- Or the **SenderResponseTimer** times out.

The *Policy Engine* **Shall** transition to the **PE_PRS_SNK_SRC_Transition_to_off** state when:

- An **Accept** Message is received.

8.3.3.19.4.8 PE_PRS_SNK_SRC_Reject_Swap State

On entry to the **PE_PRS_SNK_SRC_Reject_Swap** state the *Policy Engine* **Shall** request the *Protocol Layer* to send:

- A **Reject** Message if the device is unable to perform a *Power Role Swap* at this time.
- A **Wait** Message if further evaluation of the *Power Role Swap* request is required.

Note: In this case it is expected that one of the *Port Partners* will send a **PR_Swap** Message at a later time (see [Section 6.3.12.2, "Wait in response to a PR_Swap Message"](#)).

The *Policy Engine* **Shall** transition to the **PE_SNK_Ready** state when:

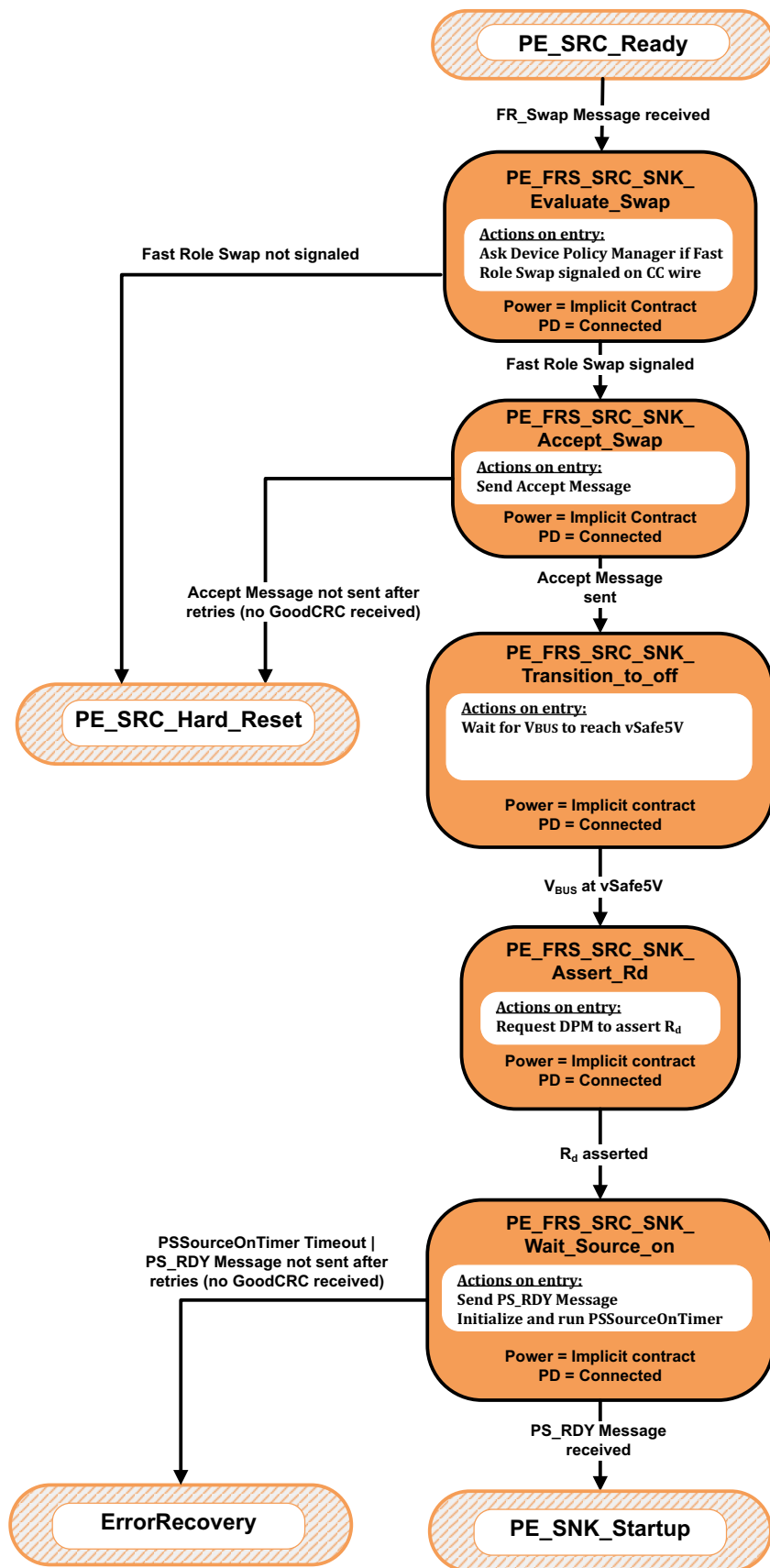
- The **Reject** or **Wait** Message has been sent.

8.3.3.19.5 Policy Engine in Source to Sink Fast Role Swap State Diagram

Dual-Role Ports that combine *Source* and *Sink* functionality **Shall** comprise *Source* and *Sink Policy Engine* state machines. In addition, they **Should** have the capability to do a *Fast Role Swap* from the **PE_SRC_Ready** state and **Shall** return to *USB Default Operation* on a *Hard Reset*.

[Figure 8.178, "Dual-Role Port in Source to Sink Fast Role Swap State Diagram"](#) shows the additional state diagram required to perform a *Fast Role Swap* from *Source* to *Sink Power Roles* and the changes that **Shall** be followed for error handling.

Figure 8.178 Dual-Role Port in Source to Sink Fast Role Swap State Diagram



8.3.3.19.5.1 PE_SRC_Ready State

The *Fast Role Swap* process **Shall** start only from the *PE_SRC_Ready* state where power is stable.

The *Policy Engine* **Shall** transition to the *PE_FRS_SRC_SNK_Evaluate_Swap* state when:

- An *FR_Swap* Message is received.

8.3.3.19.5.2 PE_FRS_SRC_SNK_Evaluate_Swap State

On entry to the *PE_FRS_SRC_SNK_Evaluate_Swap* state the *Policy Engine* **Shall** ask the *Device Policy Manager* whether *Fast Role Swap* has been signaled on the *CC* wire.

The *Policy Engine* **Shall** transition to the *PE_FRS_SRC_SNK_Accept_Swap* state when:

- The *Device Policy Manager* indicates that a *Fast Role Swap* has been signaled.

The *Policy Engine* **Shall** transition to the *PE_SRC_Hard_Reset* state when:

- The *Device Policy Manager* indicates that a *Fast Role Swap* is not being signaled.

8.3.3.19.5.3 PE_FRS_SRC_SNK_Accept_Swap State

On entry to the *PE_FRS_SRC_SNK_Accept_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send an *Accept* Message.

The *Policy Engine* **Shall** transition to the *PE_FRS_SRC_SNK_Transition_to_off* state when:

- The *Accept* Message has been sent.

The *Policy Engine* **Shall** transition to the *PE_SRC_Hard_Reset* state when:

- The *Accept* Message is not sent after retries (a *GoodCRC* Message has not been received).

Note: A *Soft Reset* **Shall Not** be initiated in this case.

8.3.3.19.5.4 PE_FRS_SRC_SNK_Transition_to_off State

On entry to the *PE_FRS_SRC_SNK_Transition_to_off* state the *Policy Engine* **Shall** wait until *V_{BUS}* has discharged to *vSafe5V*.

The *Policy Engine* **Shall** transition to the *PE_FRS_SRC_SNK_Assert_Rd* state when:

- The *Device Policy Manager* indicates that *V_{BUS}* has discharged to *vSafe5V*.

8.3.3.19.5.5 PE_FRS_SRC_SNK_Assert_Rd State

On entry to the *PE_FRS_SRC_SNK_Assert_Rd* state the *Policy Engine* **Shall** request the *Device Policy Manager* to change the resistor asserted on the *CC* wire from R_p to R_d .

The *Policy Engine* **Shall** transition to the *PE_FRS_SRC_SNK_Wait_Source_on* state when:

- The *Device Policy Manager* indicates that R_d is asserted.

8.3.3.19.5.6 PE_FRS_SRC_SNK_Wait_Source_on State

On entry to the *PE_FRS_SRC_SNK_Wait_Source_on* state the *Policy Engine* **Shall** request the *Protocol Layer* to send a *PS_RDY* Message and **Shall** start the *PSSourceOnTimer*.

On exit from the *Source off* state the *Policy Engine* **Shall** stop the *PSSourceOnTimer*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Startup* when:

- A *PS_RDY* Message is received indicating that the *New Source* is now applying R_p .

The *Policy Engine* **Shall** transition to the *ErrorRecovery* state when:

- The *PSSourceOnTimer* times out or
- The *PS_RDY* Message is not sent after retries (a *GoodCRC* Message has not been received).

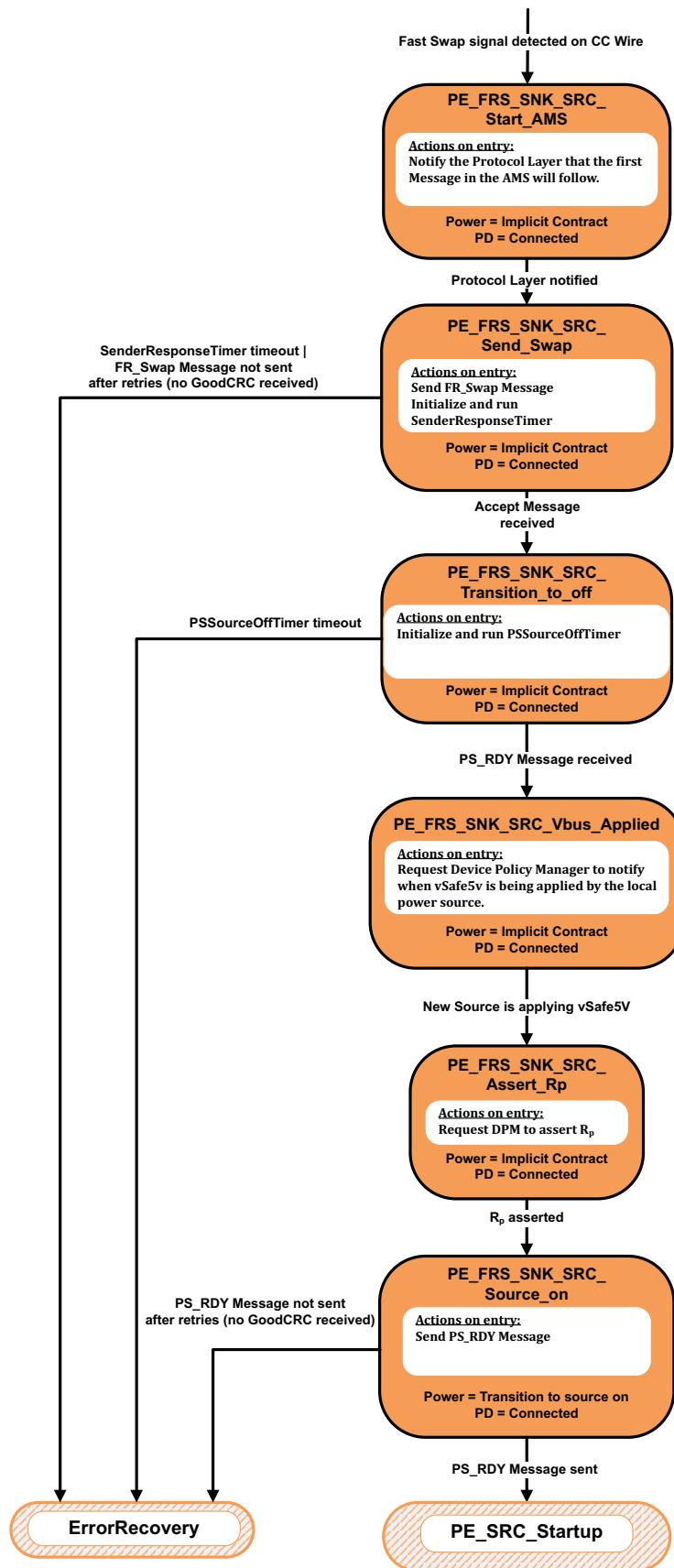
Note: A *Soft Reset* **Shall Not** be initiated in this case.

8.3.3.19.6 Policy Engine in Sink to Source Fast Role Swap State Diagram

Dual-Role Ports that combine *Sink* and *Source* functionality **Shall** comprise *Sink* and *Source Policy Engine* state machines. In addition, they **Should** have the capability to do a *Fast Role Swap* from the **PE_SNK_Ready** state and **Shall** return to *USB Default Operation* on a *Hard Reset*.

[Figure 8.179, "Dual-role Port in Sink to Source Fast Role Swap State Diagram"](#) shows the additional state diagram required to perform a *Fast Role Swap* from *Sink* to *Source Power Roles* and the changes that **Shall** be followed for error handling.

Figure 8.179 Dual-role Port in Sink to Source Fast Role Swap State Diagram



8.3.3.19.6.1 PE_FRS_SNK_SRC_Start_AMS State

The *Policy Engine* **Shall** transition to the *PE_FRS_SNK_SRC_Start_AMS* state from any other state provided there is an *Explicit Contract* in place when:

- The *Sink Capabilities* received from the *Initial Source* by the *Policy Engine* has at least one of the *Fast Role Swap* bits set.
- The system has sufficient reserve power to provide the requested current to the *Initial Source*, as requested in the *Fast Role Swap* bits in the *Sink Capabilities*, and is willing to dedicate it to the Port
- The *Device Policy Manager* indicates that a *Fast Role Swap* signal has been detected on the *CC* wire.

On entry to the *PE_FRS_SNK_SRC_Start_AMS* state the *Policy Engine* **Shall** notify the *Protocol Layer* that the first *Message* in an *AMS* will follow.

The *Policy Engine* **Shall** transition to the *PE_FRS_SNK_SRC_Send_Swap* state when:

- The *Protocol Layer* has been notified.

8.3.3.19.6.2 PE_FRS_SNK_SRC_Send_Swap State

On entry to the *PE_FRS_SNK_SRC_Send_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send an *FR_Swap* *Message* and **Shall** initialize and run the *SenderResponseTimer*.

The *Policy Engine* **Shall** transition to the *PE_FRS_SNK_SRC_Transition_to_off* state when:

- An *Accept* *Message* is received.

The *Policy Engine* **Shall** transition to the *ErrorRecovery* state when:

- The *SenderResponseTimer* times out or
- The *FR_Swap* *Message* is not sent after retries (a *GoodCRC* *Message* has not been received). A *Soft Reset* **Shall Not** be initiated in this case.

8.3.3.19.6.3 PE_FRS_SNK_SRC_Transition_to_off State

On entry to the *PE_FRS_SNK_SRC_Transition_to_off* state the *Policy Engine* **Shall** initialize and run the *PSSourceOffTimer* and then request the *Device Policy Manager* to turn off the *Sink*.

The *Policy Engine* **Shall** transition to the *ErrorRecovery* state when:

- The *PSSourceOffTimer* times out.

The *Policy Engine* **Shall** transition to the *PE_FRS_SNK_SRC_VBUS_Applied* state when:

- A *PS_RDY* *Message* is received.

8.3.3.19.6.4 PE_FRS_SNK_SRC_VBUS_Applied State

On entry to the *PE_FRS_SNK_SRC_VBUS_Applied* state the *Policy Engine* waits for a notification from the *Device Policy Manager* that the local power source has applied *vSafe5V* to *VBUS* (see [Section 5.8.6.3, "Fast Role Swap Detection"](#)).

Note: This could have already been applied prior to entering this state or could be applied while waiting in this state.

The *Policy Engine* **Shall** transition to the *PE_FRS_SNK_SRC_Assert_Rp* state when:

- The *Device Policy Manager* indicates that *vSafe5V* is being applied.

8.3.3.19.6.5 PE_FRS_SNK_SRC_Assert_Rp State

On entry to the *PE_FRS_SNK_SRC_Assert_Rp* state the *Policy Engine* **Shall** request the *Device Policy Manager* to change the resistor asserted on the *CC* wire from R_d to R_p .

The *Policy Engine* **Shall** transition to the *PE_FRS_SNK_SRC_Source_on* state when:

- The *Device Policy Manager* indicates that R_p is asserted.

8.3.3.19.6.6 PE_FRS_SNK_SRC_Source_on State

On entry to the *PE_FRS_SNK_SRC_Source_on* state the *Policy Engine* **Shall** request the *Device Policy Manager* to turn on the *Source*.

On exit from the *PE_FRS_SNK_SRC_Source_on* state the *Policy Engine* **Shall** send a *PS_RDY* Message.

The *Policy Engine* **Shall** transition to the *PE_SRC_Startup* state when:

- The *PS_RDY* Message has been sent.

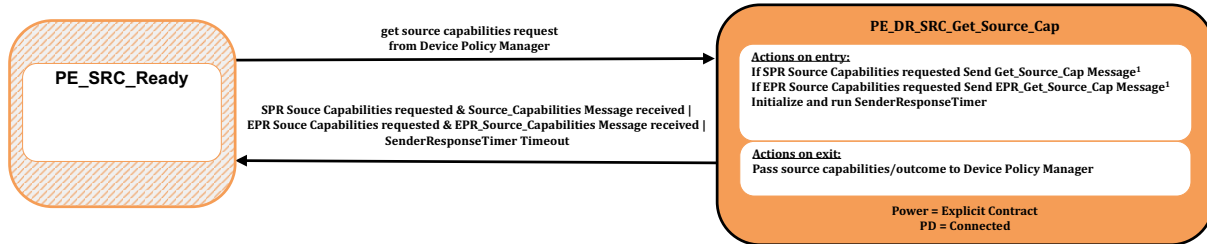
The *Policy Engine* **Shall** transition to the *ErrorRecovery* state when:

- The *PS_RDY* Message is not sent after retries (a *GoodCRC* Message has not been received). A *Soft Reset* **Shall Not** be initiated in this case.

8.3.3.19.7 Dual-Role (Source Port) Get Source Capabilities State Diagram

[Figure 8.180, "Dual-Role \(Source\) Get Source Capabilities diagram"](#) shows the state diagram for a Dual-Role device, presently operating as a *Source*, on receiving a request from the *Device Policy Manager* to get the *Port Partner's Source Capabilities*. See also [Section, "A Source Port *Shall* report its Capabilities in a series of 32-bit Power Data Objects \(see Table 6.7, "Power Data Object"\) as part of a Source Capabilities Message \(see Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"\). Power Data Objects are used to convey a Source Port's Capabilities to provide power including Dual-Role Power ports presently operating as a Sink."](#)

Figure 8.180 Dual-Role (Source) Get Source Capabilities diagram



¹⁾ Either SPR or EPR Source Capabilities *May* be requested, regardless of whether or not the Source is currently operating in SPR or EPR Mode.

8.3.3.19.7.1 PE_DR_SRC_Get_Source_Cap State

The *Policy Engine* **Shall** transition to the **PE_DR_SRC_Get_Source_Cap** state, from the **PE_SRC_Ready** state, due to a request to get the remote *Source Capabilities* from the *Device Policy Manager*.

- On entry to the **PE_DR_SRC_Get_Source_Cap** state the *Policy Engine* **Shall** request the *Protocol Layer* to send a *get Source Capabilities Message* in order to retrieve the *Source Capabilities*. The *Policy Engine* **Shall** send:
 - A **Get_Source_Cap** Message when the *Device Policy Manager* requests SPR capabilities or
 - An **EPR_Get_Source_Cap** Message when the *Device Policy Manager* requests EPR Capabilities.

The *Policy Engine* **Shall** then start the **SenderResponseTimer**.

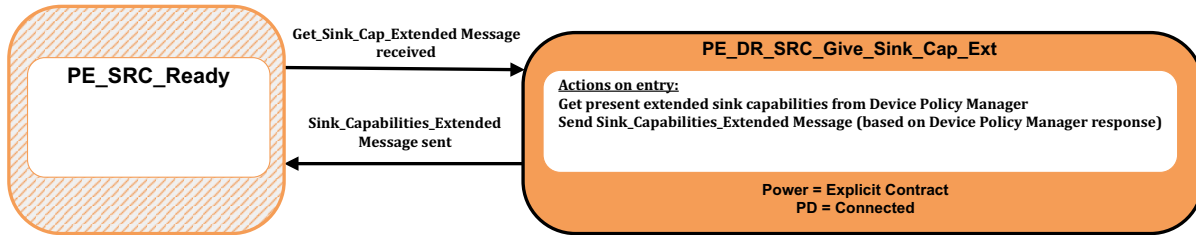
On exit from the **PE_DR_SRC_Get_Source_Cap** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** State (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- In *SPR Mode* and SPR *Source Capabilities* were requested and a **Source_Capabilities** Message is received or
- In *EPR Mode* and EPR *Source Capabilities* were requested and an **EPR_Source_Capabilities** Message is received or
- The **SenderResponseTimer** times out.

Figure 8.181, "Dual-Role (Source) Give Sink Capabilities diagram" shows the state diagram for a Dual-Role device, presently operating as a Source, on receiving a **Get_Sink_Cap** Message. See also [Section, "A Source Port Shall report its Capabilities in a series of 32-bit Power Data Objects \(see Table 6.7, "Power Data Object"\) as part of a Source_Capabilities_Message \(see Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"\). Power Data Objects are used to convey a Source Port's Capabilities to provide power including Dual-Role Power ports presently operating as a Sink."](#)

Figure 8.181 Dual-Role (Source) Give Sink Capabilities diagram



8.3.3.19.8.1

PE_DR_SRC_Give_Sink_Cap State

The Policy Engine **Shall** transition to the **PE_DR_SRC_Give_Sink_Cap** state, from the **PE_SRC_Ready** state, when a **Get_Sink_Cap** Message or **EPR_Get_Sink_Cap** Message is received.

- On entry to the **PE_DR_SRC_Give_Sink_Cap** state the Policy Engine **Shall** request the Device Policy Manager for the current system capabilities. The Policy Engine **Shall** then request the Protocol Layer to send a **Sink_Capabilities** Message containing these capabilities. The Policy Engine **Shall** send:
- A **Sink_Capabilities** Message when a **Get_Sink_Cap** Message is received or
- An **EPR_Sink_Capabilities** Message when a **EPR_Get_Sink_Cap** Message is received.

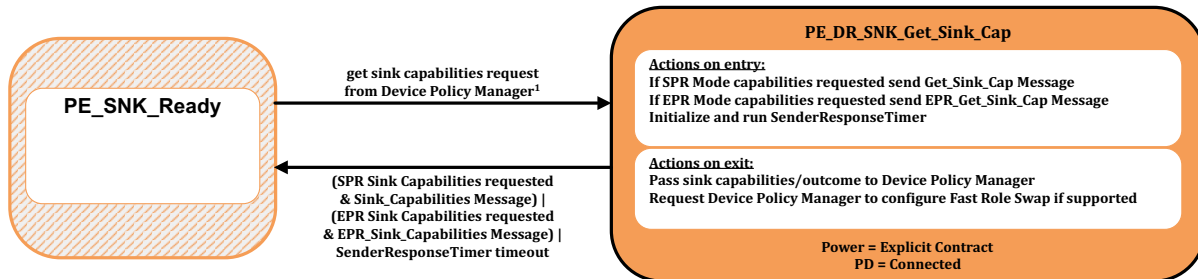
The Policy Engine **Shall** transition back to the **PE_SRC_Ready** state (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- The **Sink_Capabilities** Message has been successfully sent.

8.3.3.19.9 Dual-Role (Sink Port) Get Sink Capabilities State Diagram

Figure 8.182, "Dual-Role (Sink) Get Sink Capabilities State Diagram" shows the state diagram for a Dual-Role device, presently operating as a Sink, on receiving a request from the Device Policy Manager to get the Port Partner's Sink Capabilities. See also Section, "A Source Port **shall** report its Capabilities in a series of 32-bit Power Data Objects (see Table 6.7, "Power Data Object") as part of a Source Capabilities Message (see Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"). Power Data Objects are used to convey a Source Port's Capabilities to provide power including Dual-Role Power ports presently operating as a Sink."

Figure 8.182 Dual-Role (Sink) Get Sink Capabilities State Diagram



¹⁾ Either SPR or EPR Sink Capabilities **may** be requested, regardless of whether or not the Sink is currently operating in SPR or EPR Mode.

8.3.3.19.9.1 PE_DR_SNK_Get_Sink_Cap State

The Policy Engine **shall** transition to the **PE_DR_SNK_Get_Sink_Cap** state, from the **PE_SNK_Ready** state, due to a request to get the remote Source Capabilities from the Device Policy Manager.

- On entry to the **PE_DR_SNK_Get_Sink_Cap** state the Policy Engine **shall** request the Protocol Layer to send a **Get_Sink_Cap** Message in order to retrieve the Sink Capabilities. The Policy Engine **shall** send:
- A **Get_Sink_Cap** Message when the Device Policy Manager requests SPR capabilities or
- An **EPR_Get_Sink_Cap** Message when the Device Policy Manager requests EPR Capabilities.

The Policy Engine **shall** then start the **SenderResponseTimer**.

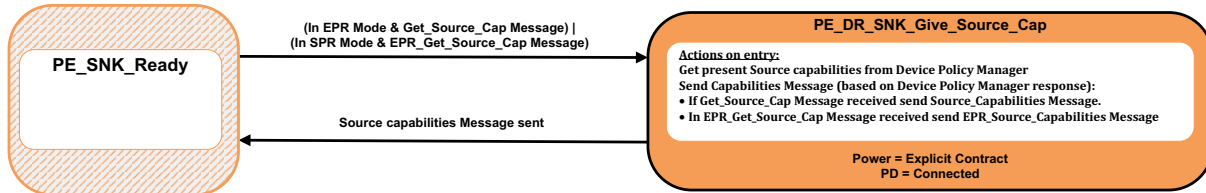
On exit from the **PE_SRC_Get_Sink_Cap** state the Policy Engine **shall** inform the Device Policy Manager of the outcome (capabilities or response timeout). If **Fast Role Swap** is supported, request Device Policy Manager prepare or disable 5V source and configure the **Fast Role Swap** receiver based on the **Fast Role Swap required USB Type-C Current** bits in the received Sink Capabilities.

The Policy Engine **shall** transition to the **PE_SNK_Ready** state (see Figure 8.133, "Sink Port State Diagram") when:

- SPR Sink Capabilities were requested and a **Sink_Capabilities** Message is received or
- EPR Sink Capabilities were requested and an **EPR_Sink_Capabilities** Message is received or
- The **SenderResponseTimer** times out.

Figure 8.182, "Dual-Role (Sink) Get Sink Capabilities State Diagram" shows the state diagram for a Dual-Role device, presently operating as a Sink, on receiving a **Get_Source_Cap** Message. See also Section, "A Source Port **Shall** report its Capabilities in a series of 32-bit Power Data Objects (see Table 6.7, "Power Data Object") as part of a **Source_Capabilities** Message (see Figure 6.13, "Example Capabilities Message with 2 Power Data Objects"). Power Data Objects are used to convey a Source Port's Capabilities to provide power including Dual-Role Power ports presently operating as a Sink."

Figure 8.183 Dual-Role (Sink) Give Source Capabilities State Diagram



8.3.3.19.10.1

PE_DR_SNK_Give_Source_Cap State

The Policy Engine **Shall** transition to the **PE_DR_SNK_Give_Source_Cap** state, from the **PE_SNK_Ready** state, when a **Get_Source_Cap** Message is received.

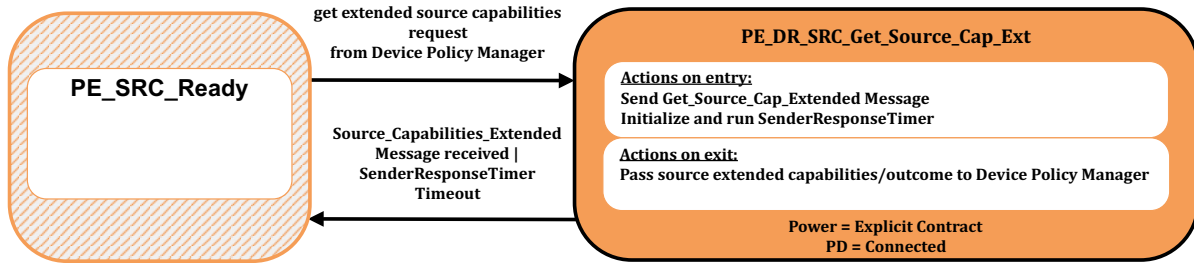
- On entry to the **PE_DR_SNK_Give_Source_Cap** State the Policy Engine **Shall** request the Device Policy Manager for the current system capabilities. The Policy Engine **Shall** then request the Protocol Layer to send a Source Capabilities Message containing these capabilities.
- The Policy Engine **Shall** send:
- A **Source_Capabilities** Message when a **Get_Source_Cap** Message is received or
- An **EPR_Source_Capabilities** Message when a **EPR_Get_Source_Cap** Message is received.

The Policy Engine **Shall** transition to the **PE_SNK_Ready** state (see Figure 8.133, "Sink Port State Diagram") when:

- The Source Capabilities Message has been successfully sent.

Figure 8.184, "Dual-Role (Source) Get Source Capabilities Extended State Diagram" shows the state diagram for a Dual-Role device, presently operating as a *Source*, on receiving a request from the *Device Policy Manager* to get the *Port Partner's* extended *Source Capabilities*. See also [Section 6.5.1, "Source Capabilities Extended Message"](#).

Figure 8.184 Dual-Role (Source) Get Source Capabilities Extended State Diagram



8.3.3.19.11.1

PE_DR_SRC_Get_Source_Cap_Ext State

The *Policy Engine* **Shall** transition to the **PE_DR_SRC_Get_Source_Cap_Ext** state, from the **PE_SRC_Ready** state, due to a request to get the remote extended *Source Capabilities* from the *Device Policy Manager*.

On entry to the **PE_DR_SRC_Get_Source_Cap_Ext** state the *Policy Engine* **Shall** send a **Get_Source_Cap_Extended Message** and initialize and run the **SenderResponseTimer**.

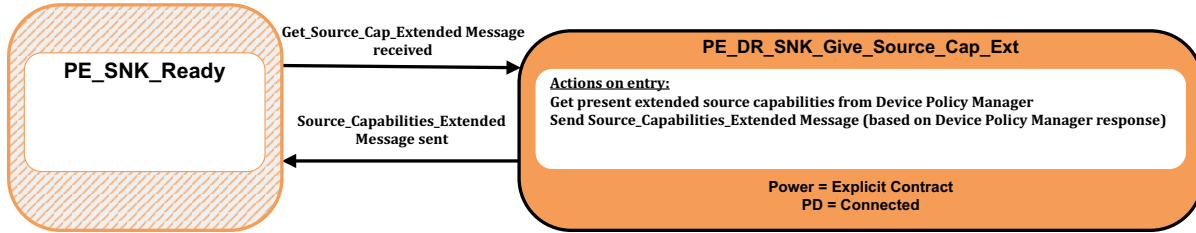
On exit from the **PE_DR_SRC_Get_Source_Cap_Ext** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** state (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- A **Source_Capabilities_Extended Message** is received
- Or **SenderResponseTimer** times out.

Figure 8.185, "Dual-Role (Sink) Give Source Capabilities Extended diagram" shows the state diagram for a Dual-Role device, presently operating as a *Sink*, on receiving a *Get_Source_Cap_Extended* Message. See also [Section 6.5.1, "Source Capabilities Extended Message"](#).

Figure 8.185 Dual-Role (Sink) Give Source Capabilities Extended diagram



8.3.3.19.12.1

PE_DR_SNK_Give_Source_Cap_Ext State

The *Policy Engine* **Shall** transition to the *PE_DR_SNK_Give_Source_Cap_Ext* state, from the *PE_SNK_Ready* state, when a *Get_Source_Cap_Extended* Message is received.

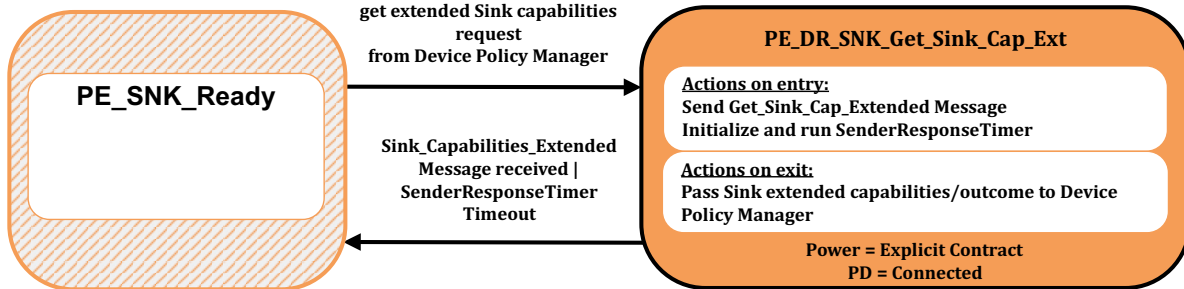
On entry to the *PE_DR_SNK_Give_Source_Cap_Ext* state the *Policy Engine* **Shall** request the present extended *Source Capabilities* from the *Device Policy Manager* and then send a *Source_Capabilities_Extended* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to the *PE_SNK_Ready* state (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- The *Source_Capabilities_Extended* Message has been successfully sent.

Figure 8.186, "Dual-Role (Sink) Get Sink Capabilities Extended State Diagram" shows the state diagram for a Dual-Role device, presently operating as a *Sink*, on receiving a request from the *Device Policy Manager* to get the *Port Partner's* extended *Sink Capabilities*. See also [Section 6.5.13, "Sink Capabilities Extended Message"](#).

Figure 8.186 Dual-Role (Sink) Get Sink Capabilities Extended State Diagram



8.3.3.19.13.1

PE_DR_SNK_Get_Sink_Cap_Ext State

The *Policy Engine* **Shall** transition to the **PE_DR_SNK_Get_Sink_Cap_Ext** state, from the **PE_SNK_Ready** state, due to a request to get the remote extended *Source Capabilities* from the *Device Policy Manager*.

On entry to the **PE_DR_SNK_Get_Sink_Cap_Ext** state the *Policy Engine* **Shall** send a **Get_Sink_Cap_Extended Message** and initialize and run the **SenderResponseTimer**.

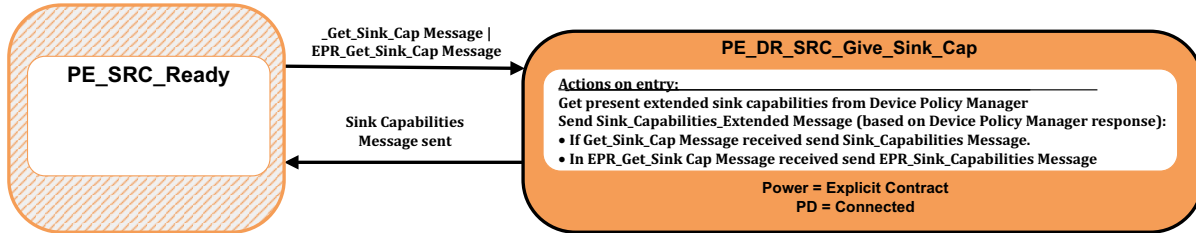
On exit from the **PE_DR_SNK_Get_Sink_Cap_Ext** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (capabilities or response timeout).

The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** state (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- A **Sink_Capabilities_Extended Message** is received.
- Or **SenderResponseTimer** times out.

Figure 8.187, "Dual-Role (Source) Give Sink Capabilities Extended diagram" shows the state diagram for a Dual-Role device, presently operating as a *Sink*, on receiving a *Get_Sink_Cap_Extended* Message. See also [Section 6.5.13](#), "Sink Capabilities Extended Message".

Figure 8.187 Dual-Role (Source) Give Sink Capabilities Extended diagram



8.3.3.19.14.1

PE_DR_SRC_Give_Sink_Cap_Ext State

The *Policy Engine* **Shall** transition to the *PE_DR_SRC_Give_Sink_Cap_Ext* state, from the *PE_SRC_Ready* state, when a *Get_Sink_Cap_Extended* Message is received.

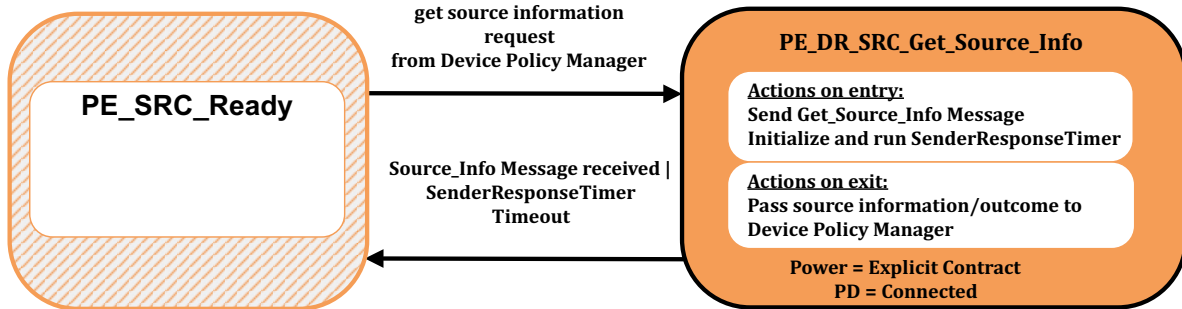
On entry to the *PE_DR_SRC_Give_Sink_Cap_Ext* state the *Policy Engine* **Shall** request the present extended *Sink Capabilities* from the *Device Policy Manager* and then send a *Sink_Capabilities_Extended* Message based on these capabilities.

The *Policy Engine* **Shall** transition back to the *PE_SRC_Ready* state (see [Figure 8.132](#), "Source Port State Diagram") when:

- The *Sink_Capabilities_Extended* Message has been successfully sent.

Figure 8.188, "Dual-Role (Source) Get Source Information State Diagram" shows the state diagram for a Dual-Role device, presently operating as a *Source*, on receiving a request from the *Device Policy Manager* to get the *Port Partner's Source* information. See also [Section 6.3.23, "Get_Source_Info Message"](#) and [Section 6.4.11, "Source_Info Message"](#).

Figure 8.188 Dual-Role (Source) Get Source Information State Diagram



8.3.3.19.15.1

PE_DR_SRC_Get_Source_Info State

The *Policy Engine* **Shall** transition to the **PE_DR_SRC_Get_Source_Info** state, from the **PE_SRC_Ready** state, due to a request to get the remote *Source* information from the *Device Policy Manager*.

On entry to the **PE_DR_SRC_Get_Source_Info** state the *Policy Engine* **Shall** send a *Get_Source_Info* Message and initialize and run the *SenderResponseTimer*.

On exit from the **PE_DR_SRC_Get_Source_Info** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the outcome (information or response timeout).

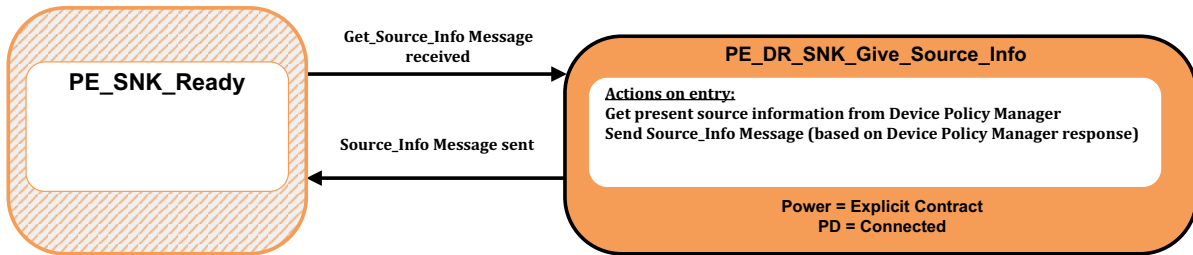
The *Policy Engine* **Shall** transition back to the **PE_SRC_Ready** state (see [Figure 8.132, "Source Port State Diagram"](#)) when:

- A *Source_Info* Message is received.
- Or *SenderResponseTimer* times out.

8.3.3.19.16 Dual-Role (Sink Port) Give Source Information State Diagram

[Figure 8.189, "Dual-Role \(Source\) Give Source Information diagram"](#) shows the state diagram for a Dual-Role device, presently operating as a *Sink*, on receiving a **Get_Source_Info** Message. See also [Section 6.3.23, "Get_Source_Info Message"](#) and [Section 6.4.11, "Source_Info Message"](#).

Figure 8.189 Dual-Role (Source) Give Source Information diagram



8.3.3.19.16.1 PE_DR_SNK_Give_Source_Info State

The *Policy Engine* **Shall** transition to the **PE_DR_SNK_Give_Source_Info** state, from the **PE_SNK_Ready** state, when a **Get_Source_Info** Message is received.

On entry to the **PE_DR_SNK_Give_Source_Info** state the *Policy Engine* **Shall** request the present *Source* information from the *Device Policy Manager* and then send a **Source_Info** Message based on this information.

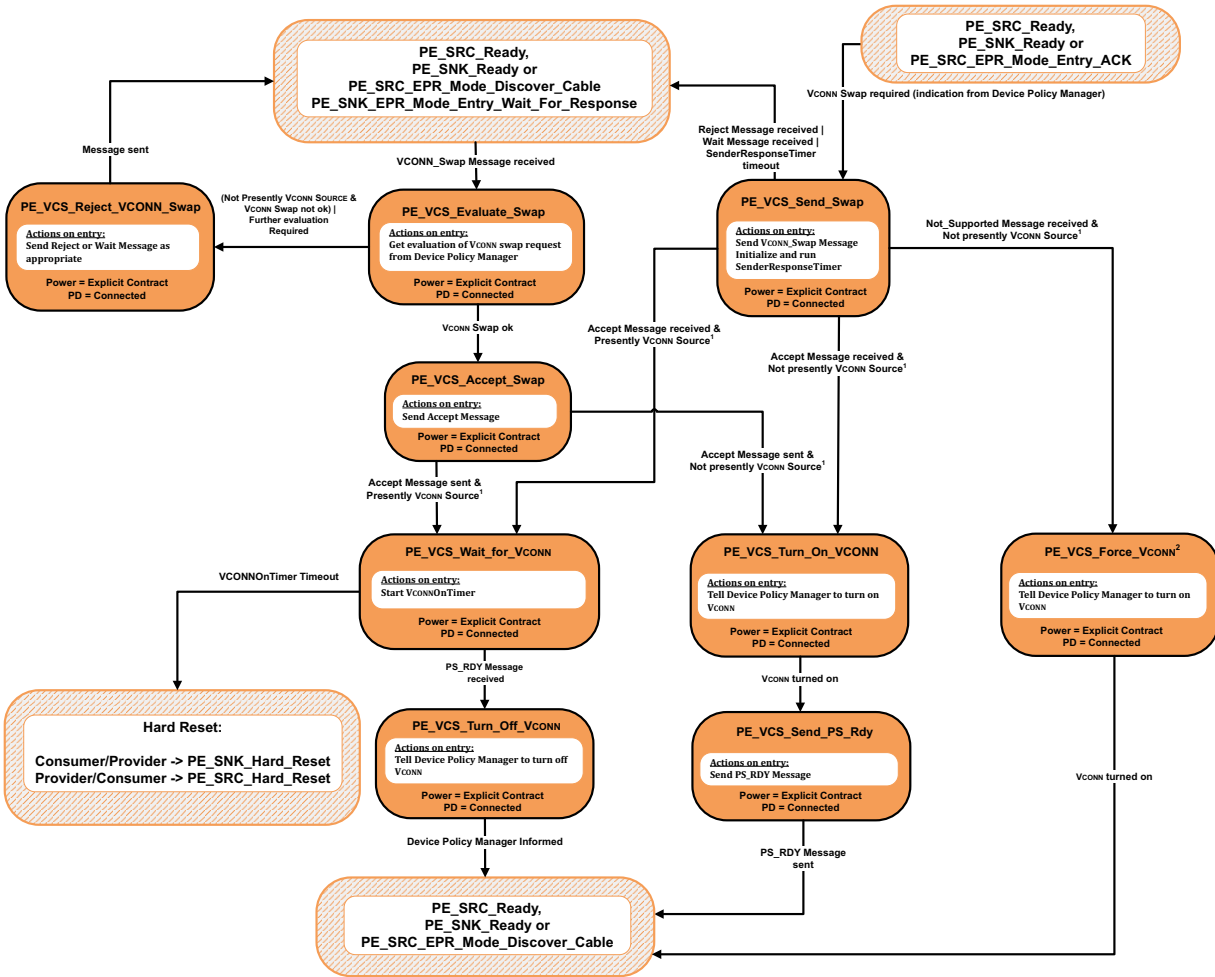
The *Policy Engine* **Shall** transition back to the **PE_SNK_Ready** state (see [Figure 8.133, "Sink Port State Diagram"](#)) when:

- The **Source_Info** Message has been successfully sent.

8.3.3.20 VCONN Swap State Diagram

The State Diagram in this section **shall** apply to Ports that supply *VCONN*. [Figure 8.190, "VCONN Swap State Diagram"](#) shows the state operation for a Port on sending or receiving a *VCONN Swap* request.

Figure 8.190 VCONN Swap State Diagram



¹ A Port is presently the VCONN Source if it has the responsibility for supplying VCONN even if VCONN has been turned off.
² The *PE_VCS_Force_VCONN* state is *Optional*.

8.3.3.20.1 PE_VCS_Send_Swap State

The *PE_VCS_Send_Swap* state is entered from either the *PE_SRC_Ready* or *PE_SNK_Ready* state when the *Policy Engine* receives a request from the *Device Policy Manager* to perform a *VCONN Swap*.

On entry to the *PE_VCS_Send_Swap* state the *Policy Engine* **shall** send a *VCONN_Swap* Message and start the *SenderResponseTimer*.

The *Policy Engine* **shall** transition to the *PE_VCS_Wait_For_VCONN* state when:

- An *Accept* Message is received and
- The Port is presently the *VCONN Source*.

The *Policy Engine* **shall** transition to the *PE_VCS_Turn_On_VCONN* state when:

- An *Accept* Message is received and
- The Port is not presently the *VCONN Source*.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_SRC_EPR_Mode_Discover_Cable* state when:

- A *Reject* Message is received or
- A *Wait* Message is received or
- The *SenderResponseTimer* times out.

The *Policy Engine* **May** transition to the *PE_VCS_Force_VCONN* state when:

- A *Not_Supported* Message is received and
- The Port is not presently the *VCONN Source*.

8.3.3.20.2 PE_VCS_Evaluate_Swap State

The *PE_VCS_Evaluate_Swap* state is entered from either the *PE_SRC_Ready* or *PE_SNK_Ready* state when the *Policy Engine* receives a *VCONN_Swap* Message.

On entry to the *PE_VCS_Evaluate_Swap* state the *Policy Engine* **Shall** request the *Device Policy Manager* for an evaluation of the *VCONN Swap* request.

The *Policy Engine* **Shall** transition to the *PE_VCS_Accept_Swap* state when:

- The *Device Policy Manager* indicates that a *VCONN Swap* is OK.

The *Policy Engine* **Shall** transition to the *PE_VCS_Reject_Swap* state when:

- The Port is not presently the *VCONN Source* and the *Device Policy Manager* indicates that a *VCONN Swap* is not OK or
- The *Device Policy Manager* indicates that a *VCONN Swap* cannot be done at this time.

8.3.3.20.3 PE_VCS_Accept_Swap State

On entry to the *PE_VCS_Accept_Swap* state the *Policy Engine* **Shall** send an *Accept* Message.

The *Policy Engine* **Shall** transition to the *PE_VCS_Wait_For_VCONN* state when:

- The *Accept* Message has been sent and
- The Port's *VCONN* is on.

The *Policy Engine* **Shall** transition to the *PE_VCS_Turn_On_VCONN* state when:

- The *Accept* Message has been sent and
- The Port's *VCONN* is off.

8.3.3.20.4 PE_VCS_Reject_Swap State

On entry to the *PE_VCS_Reject_Swap* state the *Policy Engine* **Shall** request the *Protocol Layer* to send:

- A *Reject* Message if the device is unable to perform a *VCONN Swap* at this time.
- A *Wait* Message if further evaluation of the *VCONN Swap* request is required.

Note: In this case it is expected that the Port will send a *VCONN_Swap* Message at a later time.

The *Policy Engine* **Shall** transition back to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_SRC_EPR_Mode_Discover_Cable* state when:

- The *Reject* or *Wait* Message has been sent.

8.3.3.20.5 PE_VCS_Wait_for_VCONN State

On entry to the **PE_VCS_Wait_For_VCONN** state the *Policy Engine* **Shall** start the **VCONNOnTimer**.

The *Policy Engine* **Shall** transition to the **PE_VCS_Turn_Off_VCONN** state when:

- A **PS_RDY** Message is received.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Hard_Reset** or **PE_SNK_Hard_Reset** state when:

- The **VCONNOnTimer** times out.

8.3.3.20.6 PE_VCS_Turn_Off_VCONN State

On entry to the **PE_VCS_Turn_Off_VCONN** state the *Policy Engine* **Shall** tell the *Device Policy Manager* to turn off **VCONN**.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_SRC_EPR_Mode_Discover_Cable** state when:

- The *Device Policy Manager* has been informed.

8.3.3.20.7 PE_VCS_Turn_On_VCONN State

On entry to the **PE_VCS_Turn_On_VCONN** state the *Policy Engine* **Shall** tell the *Device Policy Manager* to turn on **VCONN**.

The *Policy Engine* **Shall** transition to the **PE_VCS_Send_Ps_Rdy** state when:

- The Port's **VCONN** is on.

8.3.3.20.8 PE_VCS_Send_PS_Rdy State

On entry to the **PE_VCS_Send_Ps_Rdy** state the *Policy Engine* **Shall** send a **PS_RDY** Message.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_SRC_EPR_Mode_Discover_Cable** state when:

- The **PS_RDY** Message has been sent.

8.3.3.20.9 PE_VCS_Force_VCONN State

On entry to the **PE_VCS_Force_VCONN** state the *Policy Engine* **Shall** tell the *Device Policy Manager* to turn on **VCONN**.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_SRC_EPR_Mode_Discover_Cable** state when:

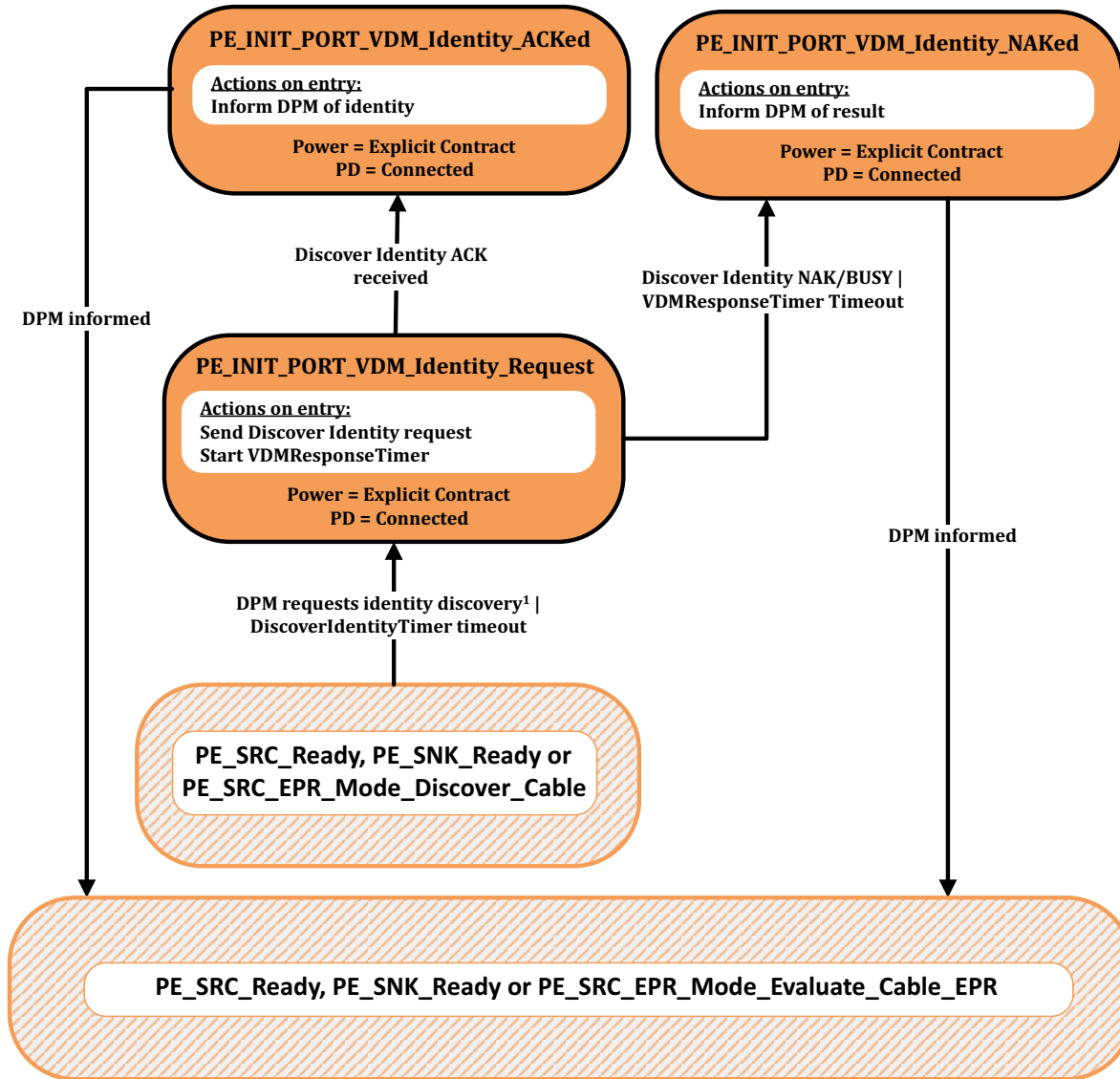
- The Port's **VCONN** is on.

8.3.3.21 Initiator Structured VDM State Diagrams

The State Diagrams in this section **Shall** apply to all *Initiators*.

Figure 8.191, "*Initiator to Port VDM Discover Identity State Diagram*" shows the state diagram for an *Initiator* when discovering the identity of its *Port Partner* or *Cable Plug*.

Figure 8.191 Initiator to Port VDM Discover Identity State Diagram



¹⁾ The DPM in an EPR Source **shall** request the discovery of the identity of the Cable Plug at startup.

8.3.3.21.1.1

PE_INIT_PORT_VDM_Identity_Request State

The *Policy Engine* transitions to the **PE_INIT_PORT_VDM_Identity_Request** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The *Device Policy Manager* requests the discovery of the identity of the *Port Partner* or *Cable Plug* or
- The **DiscoverIdentityTimer** times out.

The *Policy Engine* transitions to the **PE_INIT_PORT_VDM_Identity_Request** state from the **PE_SRC_EPR_Mode_Discover_Cable** state when:

- The *Cable Plug Discovery Process* has been initiated.

On entry to the **PE_INIT_PORT_VDM_Identity_Request** state the *Policy Engine* **Shall** send a *Structured VDM Discover Identity* Command request and **Shall** start the **VDMResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_INIT_PORT_VDM_Identity_ACKed** state when:

- A *Structured VDM Discover Identity ACK* Command response is received.

The *Policy Engine* **Shall** transition to the **PE_INIT_PORT_VDM_Identity_NAKed** state when:

- A *Structured VDM Discover Identity NAK* or **BUSY** Command response is received or
- The **VDMResponseTimer** times out.

8.3.3.21.1.2 PE_INIT_PORT_VDM_Identity_ACKed State

On entry to the **PE_INIT_PORT_VDM_Identity_ACKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the Identity information.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_SRC_EPR_Mode_Evaluate_Cable_EPR** state when:

- The *Device Policy Manager* has been informed.

8.3.3.21.1.3 PE_INIT_PORT_VDM_Identity_NAKed State

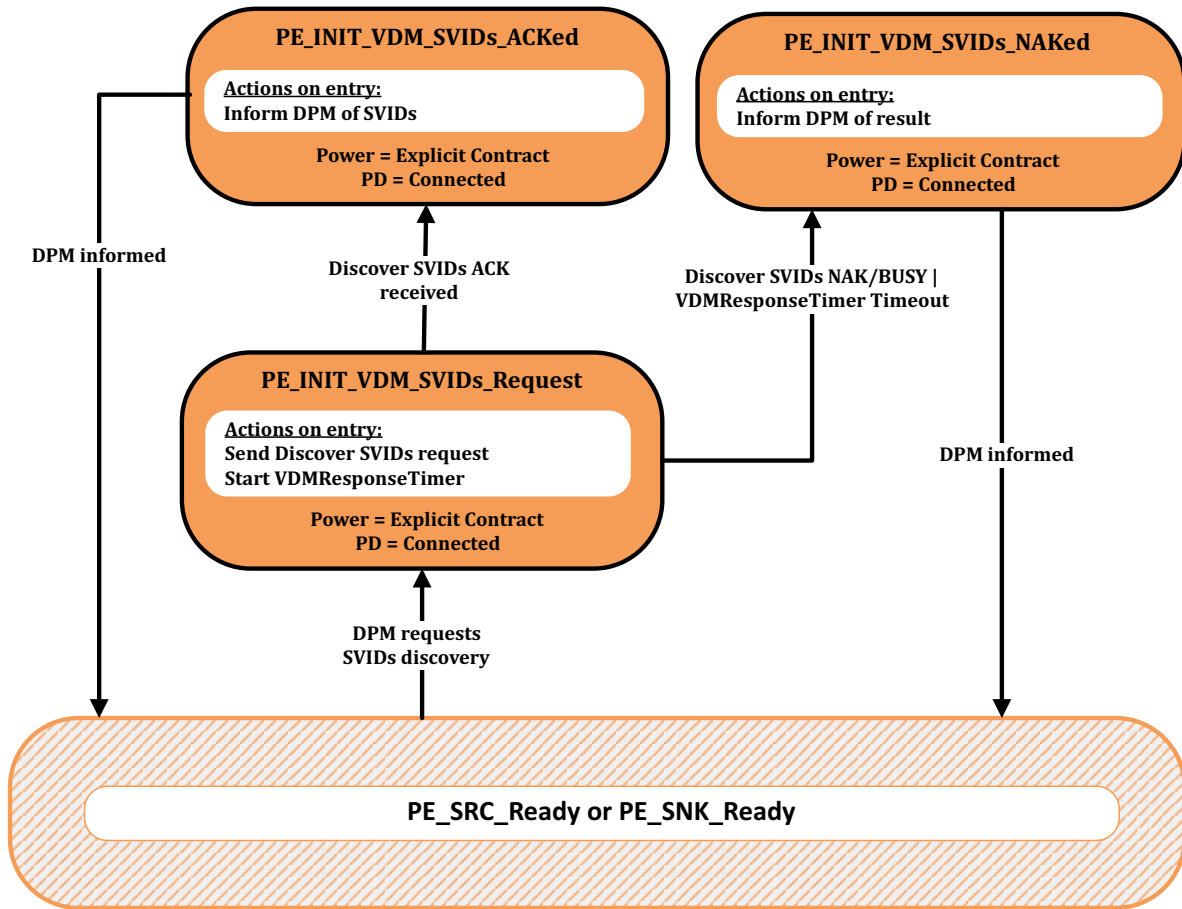
On entry to the **PE_INIT_PORT_VDM_Identity_NAKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the result (**NAK**, **BUSY** or timeout).

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_SRC_EPR_Mode_Evaluate_Cable_EPR** state when:

- The *Device Policy Manager* has been informed.

Figure 8.192, "Initiator VDM Discover SVIDs State Diagram" shows the state diagram for an Initiator when discovering SVIDs of its Port Partner or Cable Plug.

Figure 8.192 Initiator VDM Discover SVIDs State Diagram



8.3.3.21.2.1

PE_INIT_VDM_SVIDs_Request State

The Policy Engine transitions to the **PE_INIT_VDM_SVIDs_Request** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The Device Policy Manager requests the discovery of the SVIDs of the Port Partner or a Cable Plug.

On entry to the **PE_INIT_VDM_SVIDs_Request** state the Policy Engine **Shall** send a Structured VDM **Discover SVIDs** Command request and **Shall** start the **VDMResponseTimer**.

The Policy Engine **Shall** transition to the **PE_INIT_VDM_SVIDs_ACKed** state when:

- A Structured VDM **Discover SVIDs ACK** Command response is received.

The Policy Engine **Shall** transition to the **PE_INIT_VDM_SVIDs_NAKed** state when:

- A Structured VDM **Discover SVIDs NAK** or **BUSY** Command response is received or
- The **VDMResponseTimer** times out.

8.3.3.21.2.2 PE_INIT_VDM_SVIDs_ACKed State

On entry to the **PE_INIT_VDM_SVIDs_ACKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the *SVIDs* information.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The *Device Policy Manager* has been informed.

8.3.3.21.2.3 PE_INIT_VDM_SVIDs_NAKed State

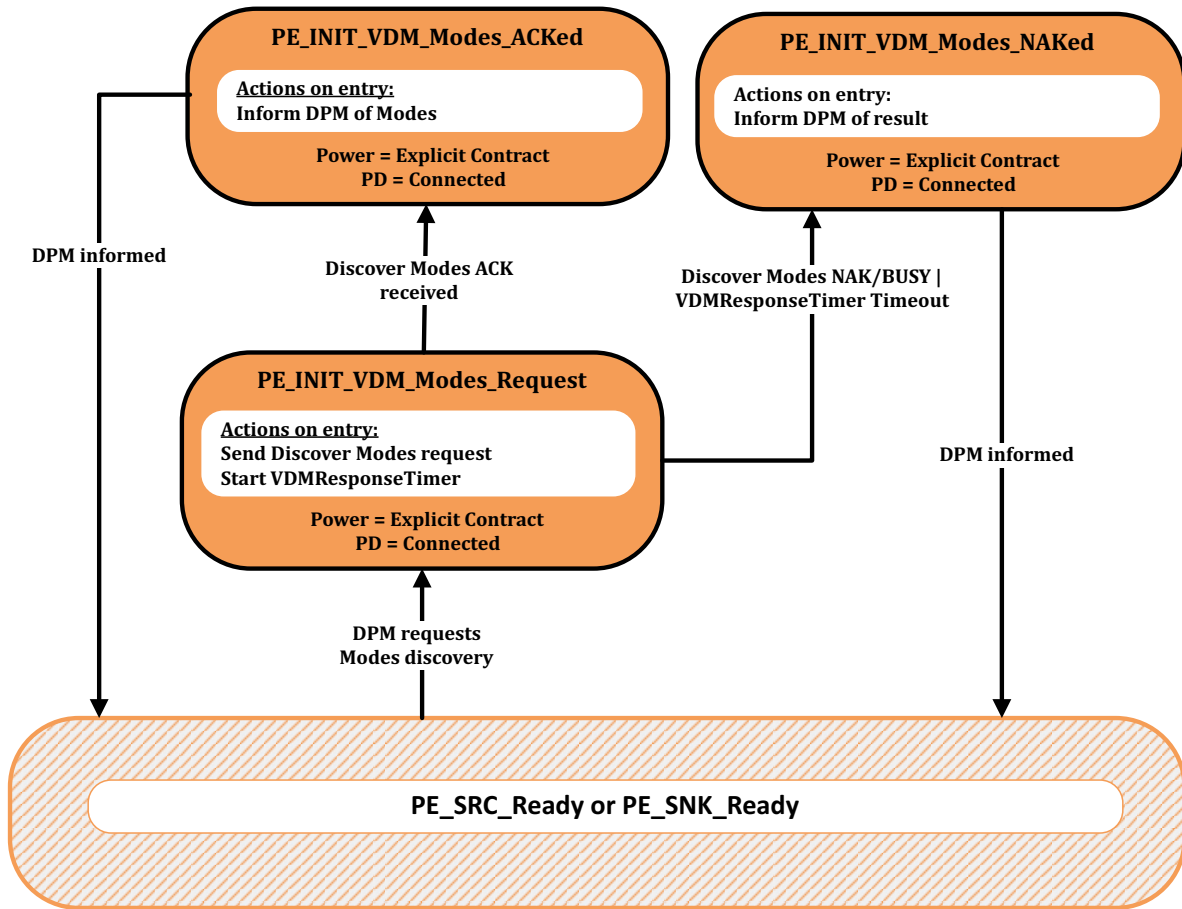
On entry to the **PE_INIT_VDM_SVIDs_NAKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the result (**NAK**, **BUSY** or timeout).

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The *Device Policy Manager* has been informed.

Figure 8.193, "Initiator VDM Discover Modes State Diagram" shows the state diagram for an *Initiator* when discovering Modes of its *Port Partner* or *Cable Plug*.

Figure 8.193 Initiator VDM Discover Modes State Diagram



8.3.3.21.3.1

PE_INIT_VDM_Modes_Request State

The *Policy Engine* transitions to the **PE_INIT_VDM_Modes_Request** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The *Device Policy Manager* requests the discovery of the Modes of the *Port Partner* or a *Cable Plug*.

On entry to the **PE_INIT_VDM_Modes_Request** state the *Policy Engine* **Shall** send a Structured VDM **Discover Modes** Command request and **Shall** start the **VDMResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_INIT_VDM_Modes_ACKed** state when:

- A Structured VDM **Discover Modes ACK** Command response is received.

The *Policy Engine* **Shall** transition to the **PE_INIT_VDM_Modes_NAKed** state when:

- A Structured VDM **Discover Modes NAK** or **BUSY** Command response is received or
- The **VDMResponseTimer** times out.

8.3.3.21.3.2 PE_INIT_VDM_Modes_ACKed State

On entry to the **PE_INIT_VDM_Modes_ACKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the Modes information.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *DFP* when:

- The *Device Policy Manager* has been informed.

8.3.3.21.3.3 PE_INIT_VDM_Modes_NAKed State

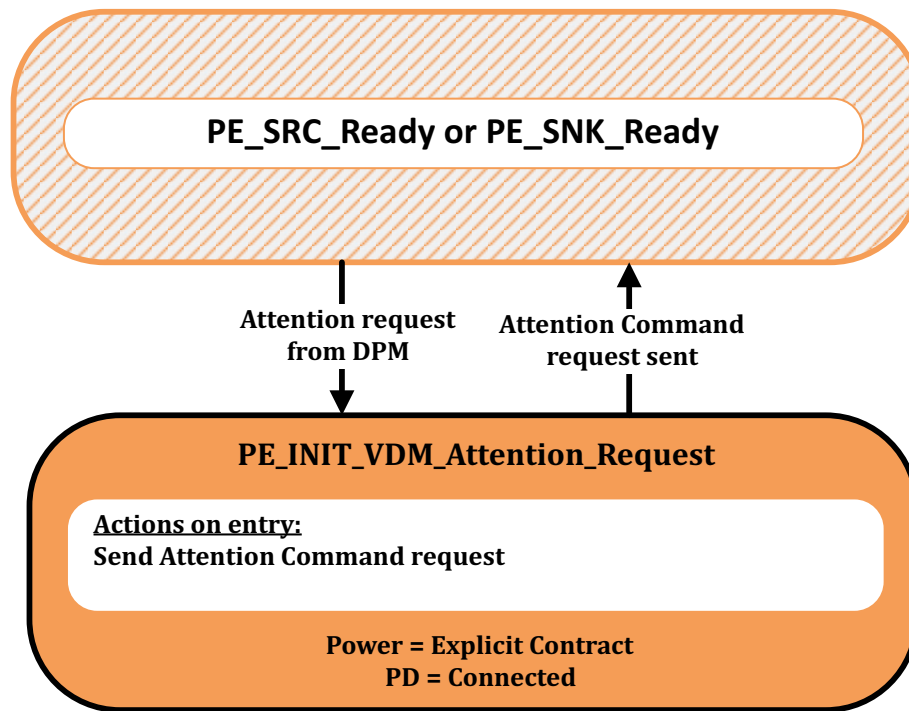
On entry to the **PE_INIT_VDM_Modes_NAKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the result (**NAK**, **BUSY** or timeout).

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *DFP* when:

- The *Device Policy Manager* has been informed.

Figure 8.194, "Initiator VDM Attention State Diagram" shows the state diagram for an *Initiator* when sending an *Attention* Command request.

Figure 8.194 Initiator VDM Attention State Diagram



8.3.3.21.4.1

PE_INIT_VDM_Attention_Request State

The *Policy Engine* transitions to the *PE_INIT_VDM_Attention_Request* state from either the *PE_SRC_Ready* or *PE_SNK_Ready* state when:

- When the *Device Policy Manager* requests attention from its *Port Partner*.

On entry to the *PE_INIT_VDM_Attention_Request* state the *Policy Engine* **Shall** send an *Attention* Command request.

The *Policy Engine* **Shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state when:

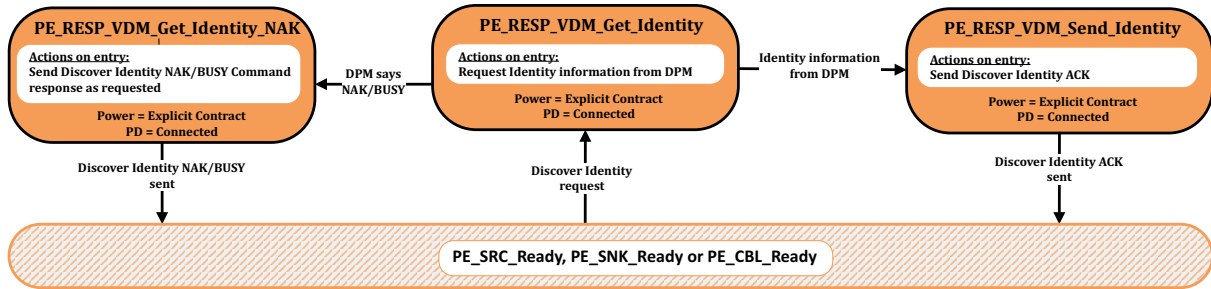
- The *Attention* Command request has been sent.

8.3.3.22 Responder Structured VDM State Diagrams

8.3.3.22.1 Responder Structured VDM Discover Identity State Diagram

Figure 8.195, "Responder Structured VDM Discover Identity State Diagram" shows the state diagram for a Responder receiving a *Discover Identity* Command request.

Figure 8.195 Responder Structured VDM Discover Identity State Diagram



8.3.3.22.1.1 PE_RESP_VDM_Get_Identity State

The Policy Engine transitions to the *PE_RESP_VDM_Get_Identity* state from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- A Structured VDM *Discover Identity* Command request is received.

On entry to the *PE_RESP_VDM_Get_Identity* state the Responder **Shall** request identity information from the Device Policy Manager.

The Policy Engine **Shall** transition to the *PE_RESP_VDM_Send_Identity* state when:

- Identity information is received from the Device Policy Manager.

The Policy Engine **Shall** transition to the *PE_RESP_VDM_Get_Identity_NAK* state when:

- The Device Policy Manager indicates that the response to the *Discover Identity* Command request is *NAK* or *BUSY*.

8.3.3.22.1.2 PE_RESP_VDM_Send_Identity State

On entry to the *PE_RESP_VDM_Send_Identity* state the Responder **Shall** send the Structured VDM *Discover Identity ACK* Command response.

The Policy Engine **Shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state for a UFP when:

- The Structured VDM *Discover Identity ACK* Command response has been sent.

8.3.3.22.1.3 PE_RESP_VDM_Get_Identity_NAK State

On entry to the *PE_RESP_VDM_Get_Identity_NAK* state the Policy Engine **Shall** send a Structured VDM *Discover Identity NAK* or *BUSY* Command response as indicated by the Device Policy Manager.

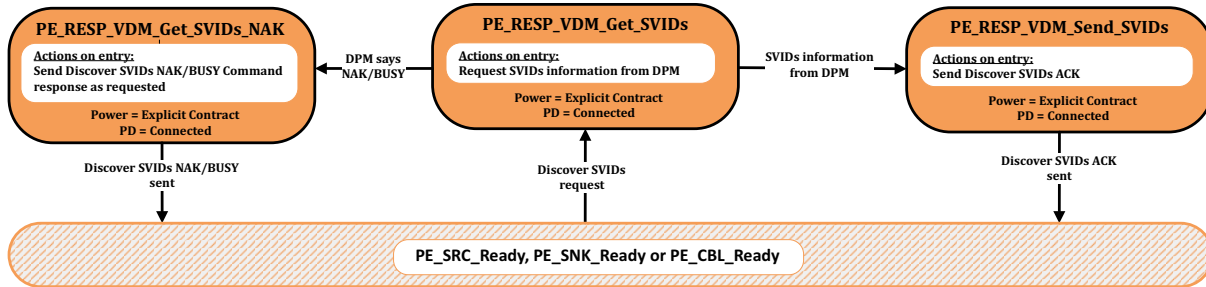
The Policy Engine **Shall** transition to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- The Structured VDM *Discover Identity NAK* or *BUSY* Command response has been sent.

8.3.3.22.2 Responder Structured VDM Discover SVIDs State Diagram

Figure 8.196, "Responder Structured VDM Discover SVIDs State Diagram" shows the state diagram for a Responder when receiving a *Discover SVIDs* Command.

Figure 8.196 Responder Structured VDM Discover SVIDs State Diagram



8.3.3.22.2.1 PE_RESP_VDM_Get_SVIDs State

The Policy Engine transitions to the *PE_RESP_VDM_Get_SVIDs* state from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- A Structured VDM *Discover SVIDs* Command request is received.

On entry to the *PE_RESP_VDM_Get_SVIDs* state the Responder **shall** request SVIDs information from the Device Policy Manager.

The Policy Engine **shall** transition to the *PE_RESP_VDM_Send_SVIDs* state when:

- SVIDs information is received from the Device Policy Manager.

The Policy Engine **shall** transition to the *PE_RESP_VDM_Get_SVIDs_NAK* state when:

- The Device Policy Manager indicates that the response to the *Discover SVIDs* Command request is *NAK* or *BUSY*.

8.3.3.22.2.2 PE_UFP_VDM_Send_SVIDs State

On entry to the *PE_RESP_VDM_Send_SVIDs* state the Responder **shall** send the Structured VDM *Discover SVIDs ACK* Command response.

The Policy Engine **shall** transition to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- The Structured VDM *Discover SVIDs ACK* Command response has been sent.

8.3.3.22.2.3 PE_UFP_VDM_Get_SVIDs_NAK State

On entry to the *PE_RESP_VDM_Get_SVIDs_NAK* state the Policy Engine **shall** send a Structured VDM *Discover SVIDs NAK* or *BUSY* Command response as indicated by the Device Policy Manager.

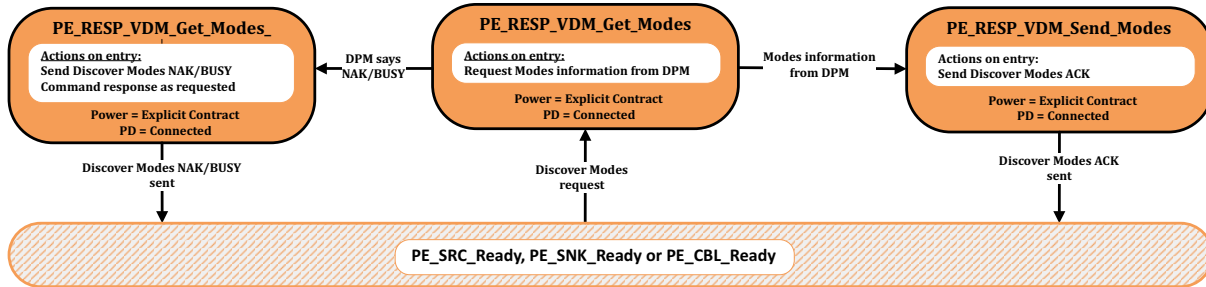
The Policy Engine **shall** transition to either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- The Structured VDM *Discover SVIDs NAK* or *BUSY* Command response has been sent.

8.3.3.22.3 Responder Structured VDM Discover Modes State Diagram

Figure 8.197, "Responder Structured VDM Discover Modes State Diagram" shows the state diagram for a Responder on receiving a **Discover Modes** Command.

Figure 8.197 Responder Structured VDM Discover Modes State Diagram



8.3.3.22.3.1 PE_RESP_VDM_Get_Modes State

The Policy Engine transitions to the **PE_RESP_VDM_Get_Modes** state from either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state when:

- A Structured VDM **Discover Modes** Command request is received.

On entry to the **PE_RESP_VDM_Get_Modes** state the Responder **Shall** request Modes information from the Device Policy Manager.

The Policy Engine **Shall** transition to the **PE_RESP_VDM_Send_Modes** state when:

- Modes information is received from the Device Policy Manager.

The Policy Engine **Shall** transition to the **PE_RESP_VDM_Get_Modes_NAK** state when:

- The Device Policy Manager indicates that the response to the **Discover Modes** Command request is **NAK** or **BUSY**.

8.3.3.22.3.2 PE_RESP_VDM_Send_Modes State

On entry to the **PE_RESP_VDM_Send_Modes** state the Responder **Shall** send the Structured VDM **Discover Modes ACK** Command response.

The Policy Engine **Shall** transition to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state when:

- The Structured VDM **Discover Modes ACK** Command response has been sent.

8.3.3.22.3.3 PE_RESP_VDM_Get_Modes_NAK State

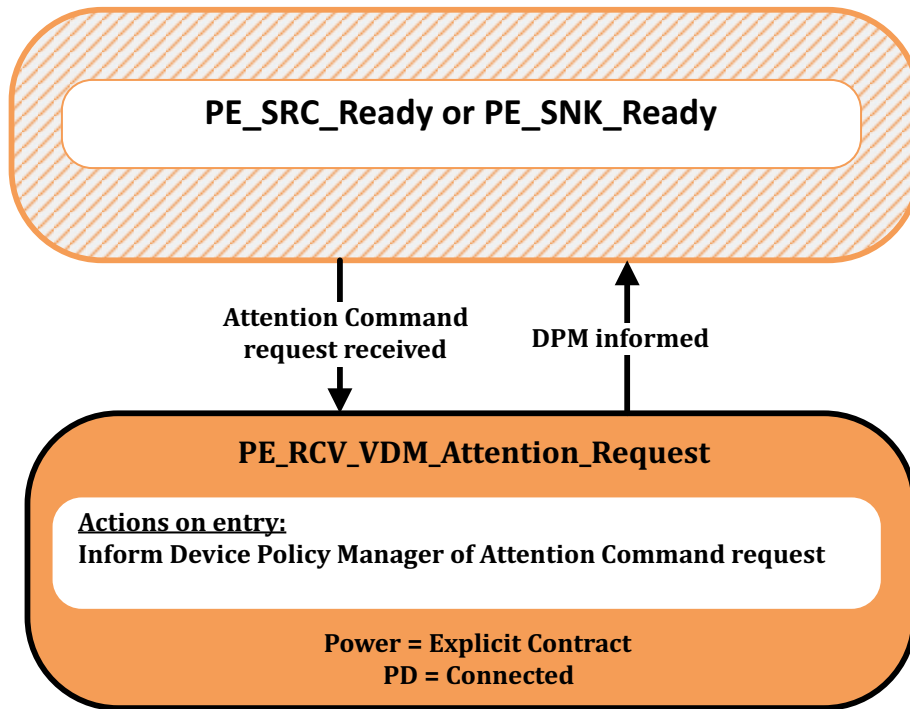
On entry to the **PE_RESP_VDM_Get_Modes_NAK** state the Policy Engine **Shall** send a Structured VDM **Discover Modes NAK** or **BUSY** Command response as indicated by the Device Policy Manager.

The Policy Engine **Shall** transition to either the **PE_SRC_Ready**, **PE_SNK_Ready** or **PE_CBL_Ready** state when:

- The Structured VDM **Discover Modes NAK** or **BUSY** Command response has been sent.

Figure 8.198, "Receiving a Structured VDM Attention State Diagram" shows the state diagram when receiving an *Attention* Command request.

Figure 8.198 Receiving a Structured VDM Attention State Diagram



8.3.3.22.4.1

PE_RCV_VDM_Attention_Request State

The *Policy Engine* transitions to the **PE_RCV_VDM_Attention_Request** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- An *Attention* Command request is received.

On entry to the **PE_RCV_VDM_Attention_Request** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the *Attention* Command request.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state when:

- The *Device Policy Manager* has been informed.

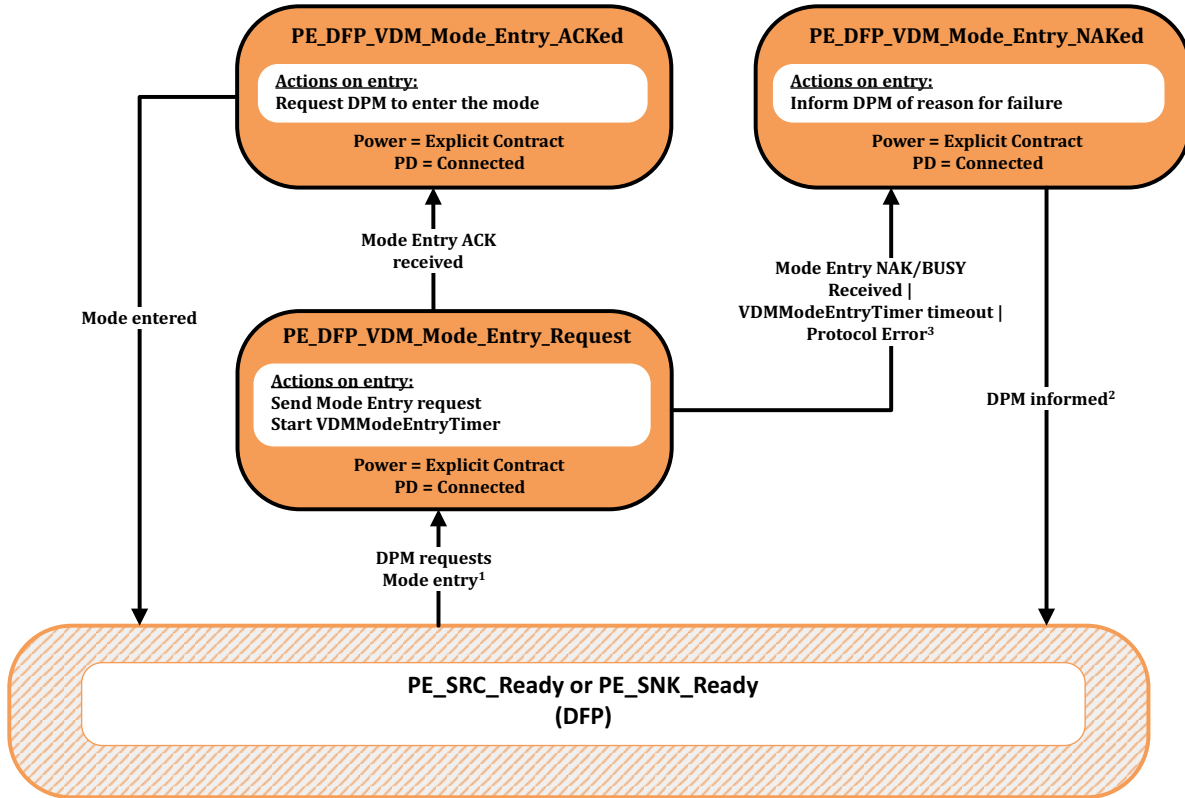
8.3.3.23 DFP Structured VDM State Diagrams

The State Diagrams in this section **shall** apply to all DFPs that support *Structured VDMs*.

8.3.3.23.1 DFP Structured VDM Mode Entry State Diagram

[Figure 8.199, "DFP VDM Mode Entry State Diagram"](#) shows the state operation for a DFP when entering a Mode.

Figure 8.199 DFP VDM Mode Entry State Diagram



- 1) The Device Policy Manager **shall** have placed the system into USB Safe State before issuing this request when entering Modal operation.
- 2) The Device Policy Manager **shall** have returned the system to USB operation if not in Modal operation at this point.
- 3) Protocol Errors are handled by informing the DPM, returning to USB Safe State and then processing the Message once the **PE_SRC_Ready** or **PE_SNK_Ready** state has been entered.

8.3.3.23.1.1 PE_DFP_VDM_Mode_Entry_Request State

The *Policy Engine* transitions to the **PE_DFP_VDM_Mode_Entry_Request** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a DFP when:

- The *Device Policy Manager* requests that the *Port Partner* or a *Cable Plug* enter a Mode.

On entry to the **PE_DFP_VDM_Mode_Entry_Request** state the *Policy Engine* **shall** send a *Structured VDM Enter Mode Command* request and **shall** start the **VDMModeEntryTimer**.

The *Policy Engine* **shall** transition to the **PE_DFP_VDM_Mode_Entry_ACKed** state when:

- A *Structured VDM Enter Mode ACK Command* response is received.

The *Policy Engine* **shall** transition to the **PE_DFP_VDM_Mode_Entry_NAKed** state when:

- A *Structured VDM Enter Mode NAK* or *BUSY Command* response is received or
- The **VDMModeEntryTimer** times out.

8.3.3.23.1.2 PE_DFP_VDM_Mode_Entry_ACKed State

On entry to the **PE_DFP_VDM_Mode_Entry_ACKed** state the *Policy Engine* **Shall** request the *Device Policy Manager* to enter the Mode.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *DFP* when:

- The Mode has been entered.

8.3.3.23.1.3 PE_DFP_VDM_Mode_Entry_NAKed State

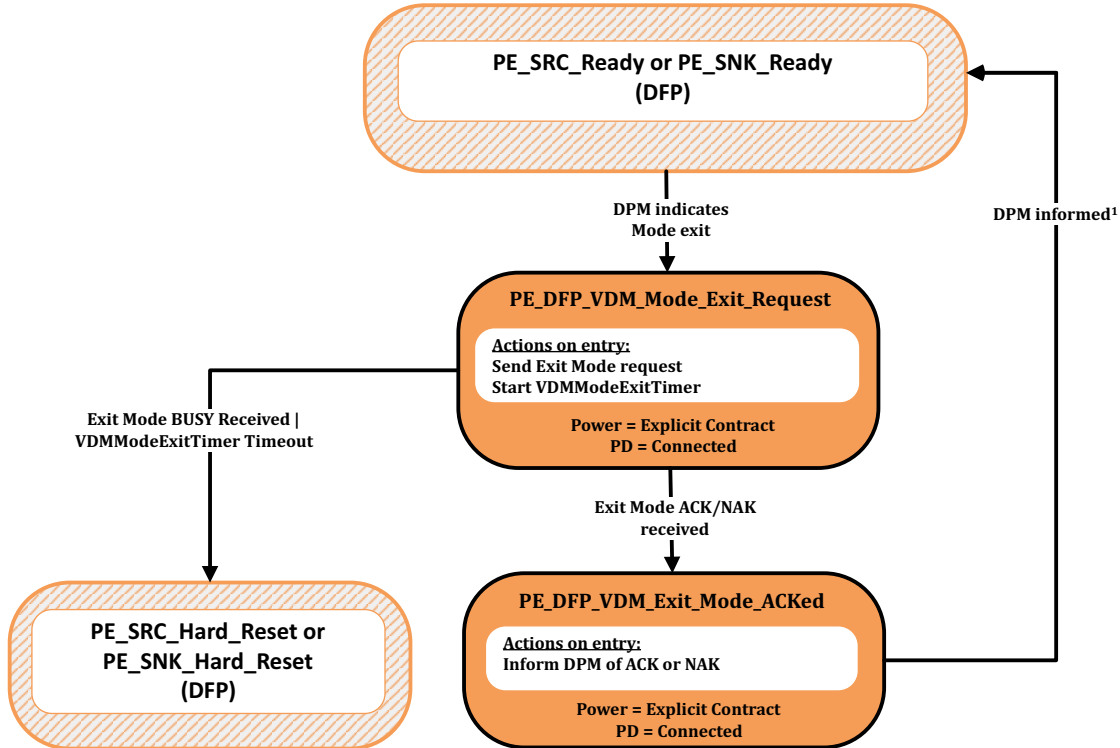
On entry to the **PE_DFP_VDM_Mode_Entry_NAKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the reason for failure (**NAK**, **BUSY**, timeout or *Protocol Error*).

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *DFP* when:

- The *Device Policy Manager* has been informed.

Figure 8.200, "DFP VDM Mode Exit State Diagram" shows the state diagram for a DFP when exiting a Mode.

Figure 8.200 DFP VDM Mode Exit State Diagram



¹⁾ The Device Policy Manager is required to return the system to USB operation at this point when exiting Modal Operation.

8.3.3.23.2.1

PE_DFP_VDM_Mode_Exit_Request State

The Policy Engine transitions to the **PE_DFP_VDM_Mode_Exit_Request** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a DFP when:

- The Device Policy Manager requests that the Port Partner or a Cable Plug exit a Mode.

On entry to the **PE_DFP_VDM_Mode_Exit_Request** state the Policy Engine **shall** send a Structured VDM **Exit Mode Command** request and **shall** start the **VDMModeExitTimer**.

The Policy Engine **shall** transition to the **PE_DFP_VDM_Mode_Entry_ACKed** state when:

- A Structured VDM **Exit Mode ACK** or **NAK** Command response is received.

The Policy Engine **shall** transition to either the **PE_SRC_Hard_Reset** or **PE_SNK_Hard_Reset** state depending on the present Power Role when:

- A Structured VDM **Exit Mode BUSY** Command response is received or
- The **VDMModeExitTimer** times out.

8.3.3.23.2.2

PE_DFP_VDM_DFP_Mode_Exit_ACKed State

On Exit to the **PE_DFP_VDM_Mode_Entry_ACKed** state the Policy Engine **shall** inform the Device Policy Manager Of the result: **ACK** or **NAK**.

The Policy Engine **shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a DFP when:

- The *Device Policy Manager* has been informed.

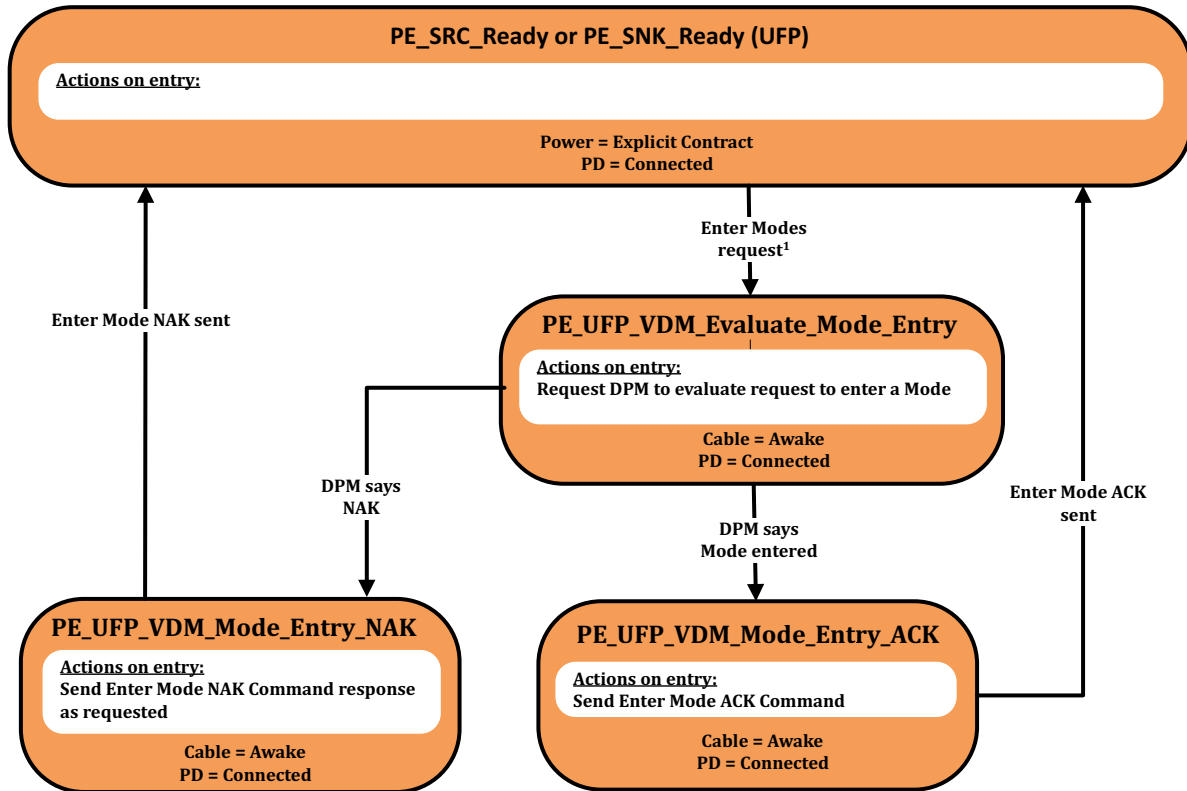
8.3.3.24 UFP Structured VDM State Diagrams

The State Diagrams in this section **shall** apply to all *UFPs* that support *Structured VDMs*.

8.3.3.24.1 UFP Structured VDM Enter Mode State Diagram

[Figure 8.201, "UFP Structured VDM Enter Mode State Diagram"](#) shows the state diagram for a *UFP* in response to an **Enter Mode** Command.

Figure 8.201 UFP Structured VDM Enter Mode State Diagram



¹) The UFP is required to be in USB operation or USB Safe State at this point.

8.3.3.24.1.1 PE_UFP_VDM_Evaluate_Mode_Entry State

The *Policy Engine* transitions to the **PE_UFP_VDM_Evaluate_Mode_Entry** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *UFP* when:

- A *Structured VDM Enter Mode* Command request is received from the *DFP*.

On Entry to the **PE_UFP_VDM_Evaluate_Mode_Entry** state the *Policy Engine* **shall** request the *Device Policy Manager* to evaluate the **Enter Mode** Command request and enter the Mode indicated in the Command request if the request is acceptable.

The *Policy Engine* **shall** transition to the **PE_UFP_VDM_Mode_Entry_ACK** state when:

- The *Device Policy Manager* indicates that the Mode has been entered.

The *Policy Engine* **shall** transition to the **PE_UFP_VDM_Mode_Entry_NAK** state when:

- The *Device Policy Manager* indicates that the response to the Mode request is **NAK**.

8.3.3.24.1.2 PE_UFP_VDM_Mode_Entry_ACK State

On entry to the **PE_UFP_VDM_Mode_Entry_ACK** state the *Policy Engine* **Shall** send a *Structured VDM Enter Mode ACK* Command response.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *UFP* when:

- The *Structured VDM Enter Mode ACK* Command response has been sent.

8.3.3.24.1.3 PE_UFP_VDM_Mode_Entry_NAK State

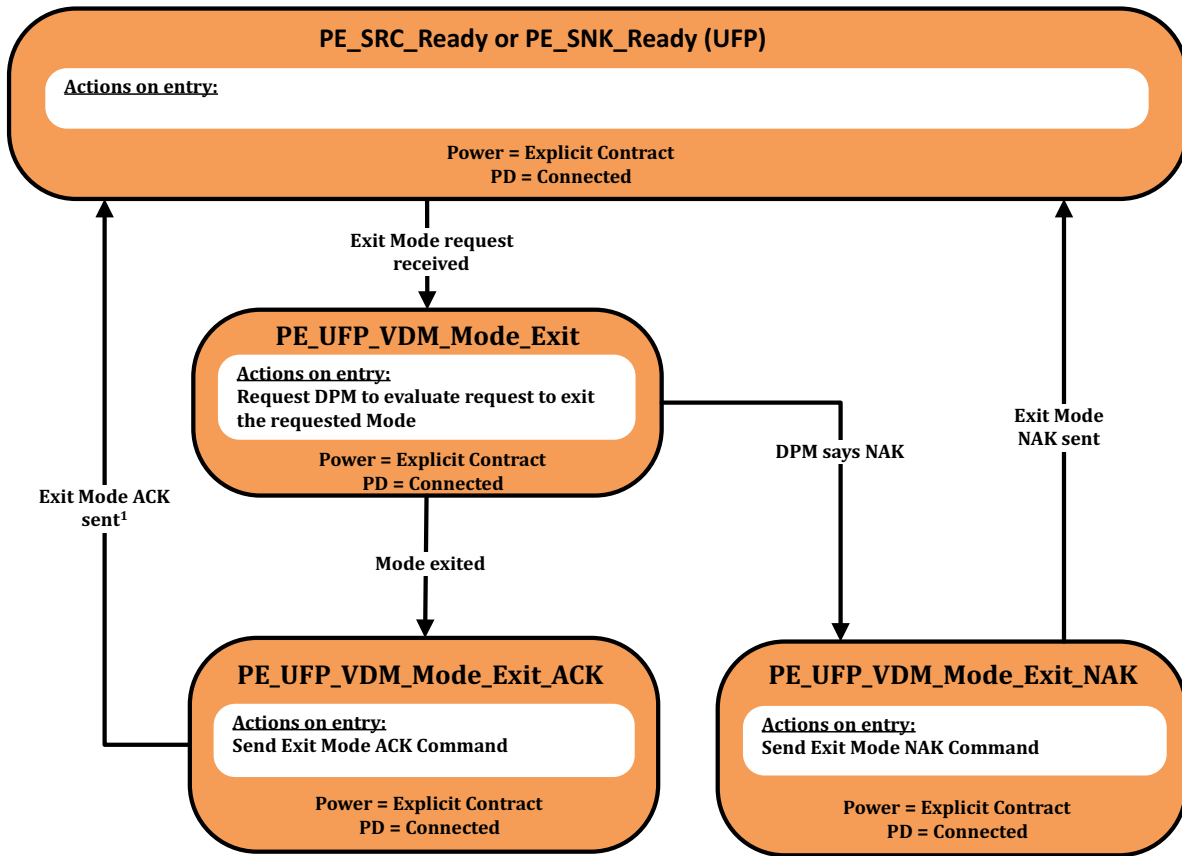
On entry to the **PE_UFP_VDM_Mode_Entry_NAK** state the *Policy Engine* **Shall** send a *Structured VDM Enter Mode NAK* Command response as indicated by the *Device Policy Manager*.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *UFP* when:

- The *Structured VDM Enter Mode NAK* Command response has been sent.

Figure 8.202, "UFP Structured VDM Exit Mode State Diagram" shows the state diagram for a UFP in response to an *Exit Mode* Command.

Figure 8.202 UFP Structured VDM Exit Mode State Diagram



1) The UFP is required to be in USB operation or USB Safe State at this point.

8.3.3.24.2.1

PE_UFP_VDM_Mode_Exit State

The *Policy Engine* transitions to the **PE_UFP_VDM_Mode_Exit** state from either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a UFP when:

- A Structured VDM *Exit Mode* Command request is received from the DFP.

On entry to the **PE_UFP_VDM_Mode_Exit** state the *Policy Engine* **Shall** request the *Device Policy Manager* to exit the Mode indicated in the Command.

The *Policy Engine* **Shall** transition to the **PE_UFP_VDM_Mode_Exit_ACK** state when:

- The *Device Policy Manager* indicates that the Mode has been exited.

The *Policy Engine* **Shall** transition to the **PE_UFP_VDM_Mode_Exit_NAK** state when:

- The *Device Policy Manager* indicates that the Command response to the *Exit Mode* Command request is **NAK**.

8.3.3.24.2.2 PE_UFP_VDM_Mode_Exit_ACK State

On entry to the **PE_UFP_VDM_Mode_Exit_ACK** state the *Policy Engine* **Shall** send a *Structured VDM Exit Mode ACK Command* response.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *UFP* when:

- The *Structured VDM Exit Mode ACK Command* response has been sent.

8.3.3.24.2.3 PE_UFP_VDM_Mode_Exit_NAK State

On entry to the **PE_UFP_VDM_Mode_Exit_NAK** state the *Policy Engine* **Shall** send a *Structured VDM Exit Mode NAK Command* response as indicated by the *Device Policy Manager*.

The *Policy Engine* **Shall** transition to either the either the **PE_SRC_Ready** or **PE_SNK_Ready** state for a *UFP* when:

- The *Structured VDM Exit Mode NAK Command* response has been sent.

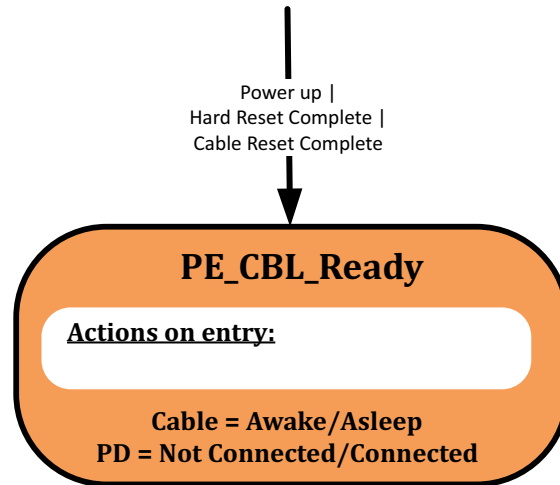
8.3.3.25 Cable Plug Specific State Diagrams

The State Diagrams in this section **shall** apply to all *Cable Plugs* that support *Structured VDMs*.

8.3.3.25.1 Cable Plug Cable Ready State Diagram

[Figure 8.203, "Cable Ready State Diagram"](#) shows the Cable Ready state diagram for a *Cable Plug*.

Figure 8.203 Cable Ready State Diagram



8.3.3.25.1.1 PE_CBL_Ready State

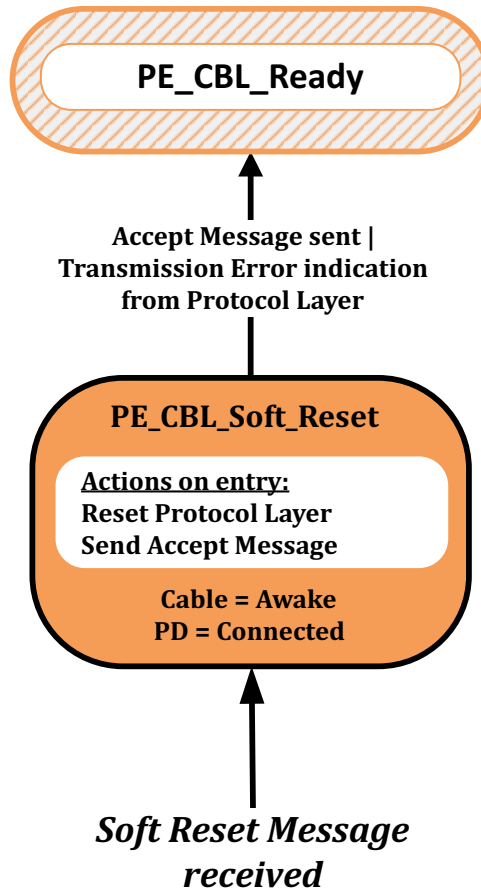
The **PE_CBL_Ready** state shown in the following sections is the normal operational state for a *Cable Plug* and where it starts after power up or a Hard/*Cable Reset*.

8.3.3.25.2 Soft/Hard/Cable Reset

8.3.3.25.2.1 Cable Plug Soft Reset State Diagram

[Figure 8.204, "Cable Plug Soft Reset State Diagram"](#) shows the *Cable Plug* state diagram on reception of a *Soft_Reset* Message.

Figure 8.204 Cable Plug Soft Reset State Diagram



8.3.3.25.2.1.1 PE_CBL_Soft_Reset State

The **PE_CBL_Soft_Reset** state **Shall** be entered from any state when a *Soft_Reset* Message is received from the *Protocol Layer*.

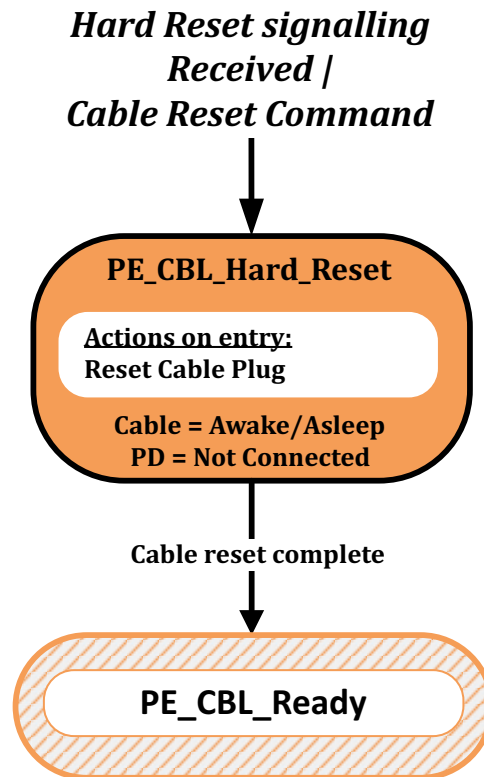
On entry to the **PE_CBL_Soft_Reset** state the *Policy Engine* **Shall** reset the *Protocol Layer* in the *Cable Plug* and **Shall** then request the *Protocol Layer* to send an *Accept* Message.

The *Policy Engine* **Shall** transition to the **PE_CBL_Ready** state when:

- The *Accept* Message has been sent or
- The *Protocol Layer* indicates that a transmission error has occurred.

[Figure 8.205, "Cable Plug Hard Reset State Diagram"](#) shows the *Cable Plug* state diagram for a *Hard Reset* or *Cable Reset*.

Figure 8.205 Cable Plug Hard Reset State Diagram



8.3.3.25.2.2.1

PE_CBL_Hard_Reset State

The **PE_CBL_Hard_Reset** state **shall** be entered from any state when either **Hard Reset Signaling** or **Cable Reset Signaling** is detected.

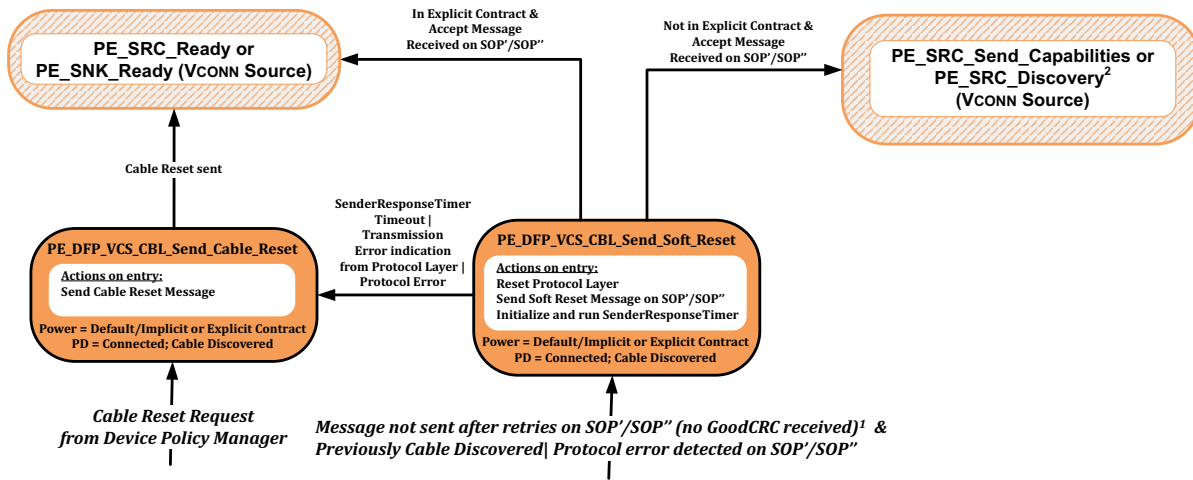
On entry to the **PE_CBL_Hard_Reset** state the *Policy Engine* **shall** reset the *Cable Plug* (equivalent to a power cycle).

The *Policy Engine* **shall** transition to the **PE_CBL_Ready** state when:

- The *Cable Plug* reset is complete.

Figure 8.206, "DFP/VCONN Source Soft Reset or Cable Reset of a Cable Plug or VPD State Diagram" below shows the state diagram for the *Policy Engine* in a *VCONN Source* when performing a *Soft Reset* or *Cable Reset* of a *Cable Plug* or *VPD* on *SOP'/SOP"*. The following sections describe operation in each of the states.

Figure 8.206 DFP/VCONN Source Soft Reset or Cable Reset of a Cable Plug or VPD State Diagram



1) Excludes the *Soft_Reset* Message itself.

2) Sink only communicates with the Cable Plug when in an Explicit Contract. If the *Discover Identity* Command is being sent at startup, then the Policy Engine will subsequently transition to the *PE_SRC_Send_Capabilities* state as normal. Otherwise, the Policy Engine will transition to the *PE_SRC_Discovery* state.

8.3.3.25.2.3.1

PE_DFP_VCS_CBL_Send_Soft_Reset State

The *PE_DFP_VCS_CBL_Send_Soft_Reset* state **shall** be entered from any state when a *Protocol Error* is detected on *SOP'/SOP"* by the *Protocol Layer* (see [Section 6.8.1, "Soft Reset and Protocol Error"](#)) or when a *Message* has not been sent after retries on *SOP'/SOP"* while communicating with a *Cable Plug/VPD* and when there was previous communication with the *Cable Plug* that did not result in a *Transmission Error* or whenever the *Device Policy Manager* directs a *Soft Reset* on *SOP'/SOP"*.

On entry to the *PE_DFP_VCS_CBL_Send_Soft_Reset* state the *DFP Policy Engine* **shall** request the *SOP'/SOP"* *Protocol Layer* to perform a *Soft Reset*, then **shall** send a *Soft_Reset* Message on *SOP'/SOP"* to the *Cable Plug/VPD*, and initialize and run the *SenderResponseTimer*.

The *Policy Engine* **shall** transition to either the *PE_SRC_Ready* or *PE_SNK_Ready* state, depending on the *DFP VCONN Source's Power Role*, when:

- There is no *Explicit Contract* in place and
- An *Accept* Message has been received on *SOP'/SOP"*.

The *Policy Engine* **shall** transition to either the *PE_SRC_Send_Capabilities* state or *PE_SRC_Discovery* state, depending on the *DFP's VCONN Source's Power Role*, when:

- There is an *Explicit Contract* in place and
- An *Accept* Message has been received on *SOP'/SOP"*.

The *Policy Engine* **shall** transition to the *PE_DFP_VCS_CBL_Send_Cable_Reset* state when:

- A *SenderResponseTimer* timeout occurs
- Or the *Protocol Layer* indicates that a transmission error has occurred
- Or when a *Protocol Error* is detected on *SOP'/SOP"* by the *Protocol Layer*.

The **PE_DFP_VCS_CBL_Send_Cable_Reset** state **Shall** be entered from any state when the *Device Policy Manager* requests a *Cable Reset*.

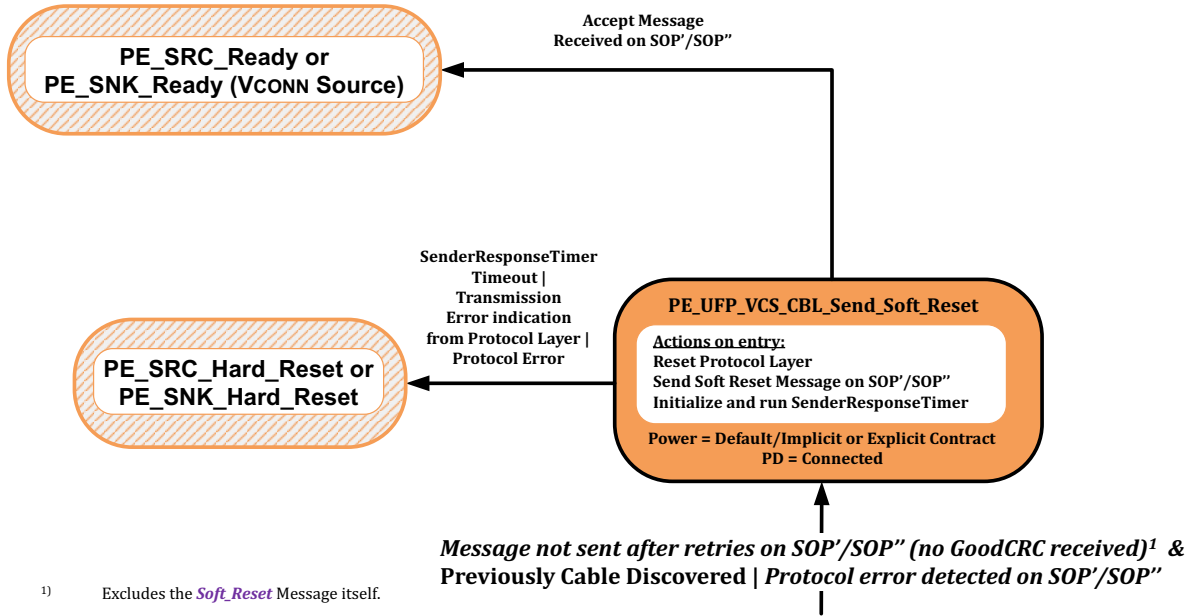
On entry to the **PE_DFP_VCS_CBL_Send_Cable_Reset** state the *DFP Policy Engine* **Shall** request the *Protocol Layer* to send **Cable Reset** Signaling.

The *Policy Engine* **Shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state, depending on the *VCONN Source's Power Role*, when:

- **Cable Reset** Signaling has been sent.

Figure 8.207, "UFP/VCONN Source Soft Reset of a Cable Plug or VPD State Diagram" below shows the state diagram for the UFP Policy Engine in a VCONN Source when performing a Soft Reset of a Cable Plug or VPD on SOP'/SOP''. The following sections describe operation in each of the states.

Figure 8.207 UFP/VCONN Source Soft Reset of a Cable Plug or VPD State Diagram



8.3.3.25.2.4.1

PE_UFP_VCS_CBL_Send_Soft_Reset State

The **PE_UFP_VCS_CBL_Send_Soft_Reset** state **shall** be entered from any state when a *Protocol Error* is detected on SOP'/SOP'' by the Protocol Layer (see [Section 6.8.1, "Soft Reset and Protocol Error"](#)) or when a *Message* has not been sent after retries on SOP'/SOP'' while communicating with a *Cable Plug*/VPD and when there was previous communication with the *Cable Plug* that did not result in a *Transmission Error* or whenever the *Device Policy Manager* directs a *Soft Reset* on SOP'/SOP''.

On entry to the **PE_UFP_VCS_CBL_Send_Soft_Reset** state the *Policy Engine* **shall** request the SOP'/SOP'' *Protocol Layer* to perform a *Soft Reset*, then **shall** send a *Soft_Reset* Message on SOP'/SOP'' to the *Cable Plug*, and initialize and run the *SenderResponseTimer*.

The *Policy Engine* **shall** transition to either the **PE_SRC_Ready** or **PE_SNK_Ready** state, depending on the UFP VCONN Source's *Power Role*, when:

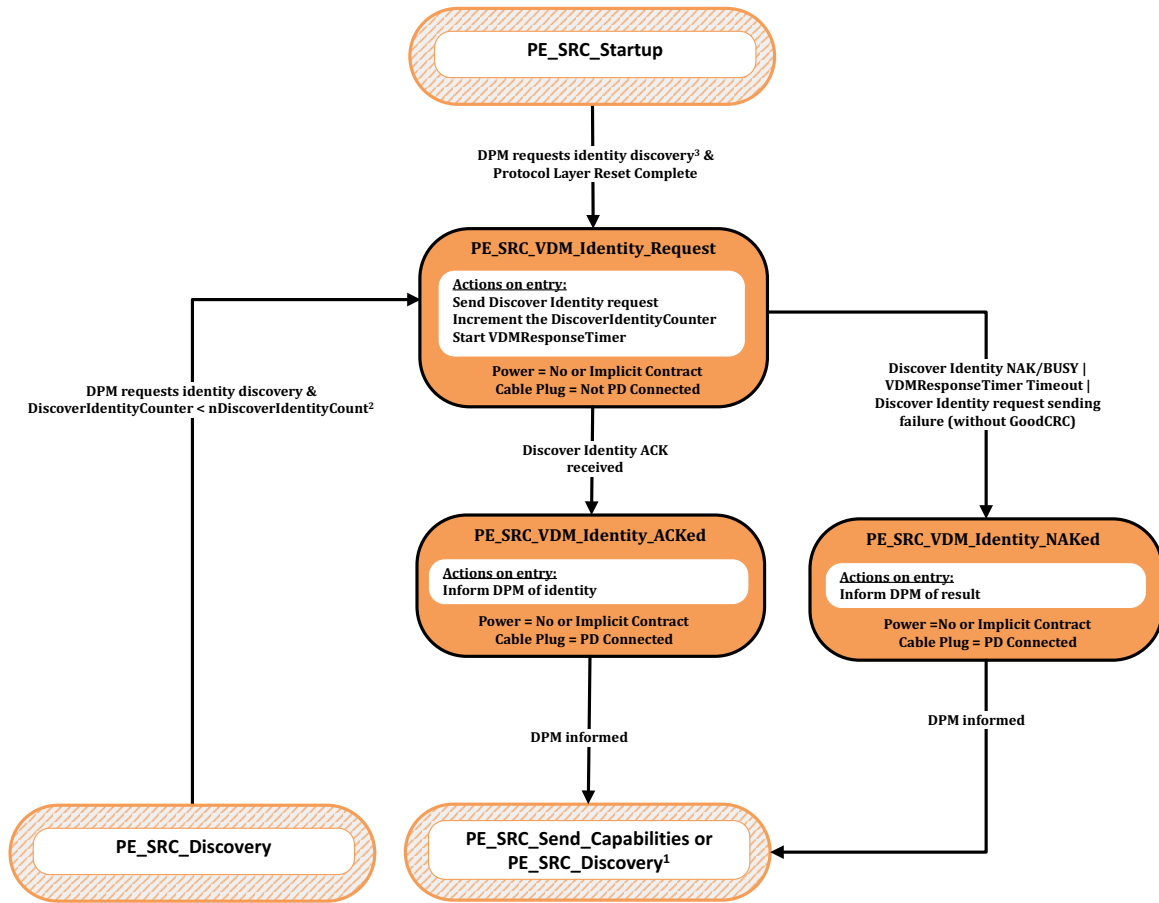
- An *Accept* Message has been received on SOP'/SOP''.

The *Policy Engine* **shall** transition to either the **PE_SRC_Hard_Reset** or **PE_SNK_Hard_Reset** state, depending on the UFP VCONN Source's *Power Role*, when:

- A *SenderResponseTimer* timeout occurs
- Or the *Protocol Layer* indicates that a transmission error has occurred
- Or when a *Protocol Error* is detected on SOP'/SOP'' by the *Protocol Layer*.

Figure 8.208, "Source Startup Structured VDM Discover Identity State Diagram" shows the state diagram for Source discovery of identity information from a Cable Plug during the startup sequence.

Figure 8.208 Source Startup Structured VDM Discover Identity State Diagram



- 1) If the *Discover Identity* Command is being sent at startup, then the Policy Engine will subsequently transition to the *PE_SRC_Send_Capabilities* state as normal. Otherwise, the Policy Engine will transition to the *PE_SRC_Discovery* state.
- 2) The *SourceCapabilityTimer* continues to run during the states defined in this diagram even though there has been an exit from the *PE_SRC_Discovery* state. This ensures that *Source_Capabilities* Messages are sent out at a regular rate.
- 3) The DPM in an EPR Source Shall request the discovery of the identity of the Cable Plug at startup.

8.3.3.25.3.1

PE_SRC_VDM_Identity_Request State

The Policy Engine **Shall** transition to the *PE_SRC_VDM_Identity_Request* state from the *PE_SRC_Startup* state when:

- The Device Policy Manager requests the discovery of the identity of the Cable Plug.

The Policy Engine **Shall** transition to the *PE_SRC_VDM_Identity_Request* state from the *PE_SRC_Discovery* state when:

- The Device Policy Manager requests the discovery of the identity of the Cable Plug and
- The *DiscoverIdentityCounter* < *nDiscoverIdentityCount*.

Even though there has been a transition out of the *PE_SRC_Discovery* state the *SourceCapabilityTimer* **Shall** continue to run during the states shown in Figure 8.208, "Source Startup Structured VDM Discover Identity State Diagram" and **Shall Not** be initialized on re-entry to *PE_SRC_Discovery*.

Note: An *EPR Source* is required to discover the identity of the *Cable Plug* prior to entering the *First Explicit Contract* (see [Section 6.4.10.1, "Process to enter EPR Mode"](#))

On entry to the **PE_SRC_VDM_Identity_Request** state the *Policy Engine* **Shall** send a *Structured VDM Discover Identity* Command request, **Shall** increment the **DiscoverIdentityCounter** and **Shall** start the **VDMResponseTimer**.

The *Policy Engine* **Shall** transition to the **PE_SRC_VDM_Identity_ACKed** state when:

- A *Structured VDM Discover Identity ACK* Command response is received.

The *Policy Engine* **Shall** transition to the **PE_SRC_VDM_Identity_NAKed** state when:

- A *Structured VDM Discover Identity NAK* or *BUSY* Command response is received or
- The **VDMResponseTimer** times out or
- The *Structured VDM Discover Identity* Command request *Message* sending fails (no **GoodCRC** Message received after retries).

8.3.3.25.3.2 PE_SRC_VDM_Identity_ACKed State

On entry to the **PE_SRC_VDM_Identity_ACKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the Identity information.

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Send_Capabilities** or **PE_SRC_Discovery** state when:

- The *Device Policy Manager* has been informed.

8.3.3.25.3.3 PE_SRC_VDM_Identity_NAKed State

On entry to the **PE_SRC_VDM_Identity_NAKed** state the *Policy Engine* **Shall** inform the *Device Policy Manager* of the result (**NAK**, **BUSY** or timeout).

The *Policy Engine* **Shall** transition back to either the **PE_SRC_Send_Capabilities** or **PE_SRC_Discovery** state when:

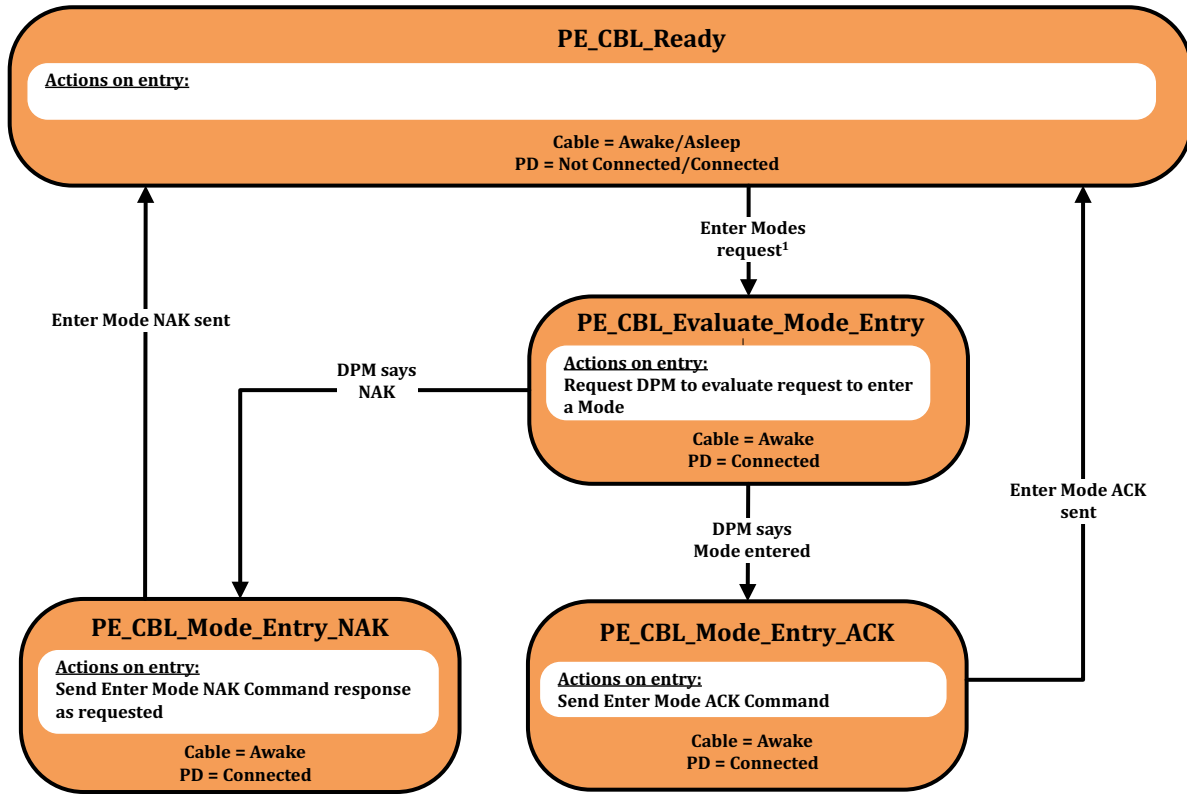
- The *Device Policy Manager* has been informed.

8.3.3.25.4 Cable Plug Mode Entry/Exit

8.3.3.25.4.1 Cable Plug Structured VDM Enter Mode State Diagram

[Figure 8.209, "Cable Plug Structured VDM Enter Mode State Diagram"](#) shows the state diagram for a Cable Plug in response to an **Enter Mode** Command.

Figure 8.209 Cable Plug Structured VDM Enter Mode State Diagram



¹⁾ The Cable is required to be in USB operation or USB Safe State at this point.

8.3.3.25.4.1.1 PE_CBL_Evaluate_Mode_Entry State

The Policy Engine transitions to the **PE_CBL_Evaluate_Mode_Entry** state from the **PE_CBL_Ready** state when:

- A Structured VDM **Enter Mode** Command request is received from the DFP.

On Entry to the **PE_CBL_Evaluate_Mode_Entry** state the Policy Engine **shall** request the Device Policy Manager to evaluate the **Enter Mode** Command request and enter the Mode indicated in the Command request if the request is acceptable.

The Policy Engine **shall** transition to the **PE_CBL_Mode_Entry_ACK** state when:

- The Device Policy Manager indicates that the Mode has been entered.

The Policy Engine **shall** transition to the **PE_CBL_Mode_Entry_NAK** state when:

- The Device Policy Manager indicates that the response to the Mode request is **NAK**.

8.3.3.25.4.1.2 PE_CBL_Mode_Entry_ACK State

On entry to the **PE_CBL_Mode_Entry_ACK** state the Policy Engine **shall** send a Structured VDM **Enter Mode ACK** Command response.

The *Policy Engine* **Shall** transition to the *PE_CBL_Ready* state when:

- The *Structured VDM Enter Mode ACK* Command response has been sent.

8.3.3.25.4.1.3 PE_CBL_Mode_Entry_NAK State

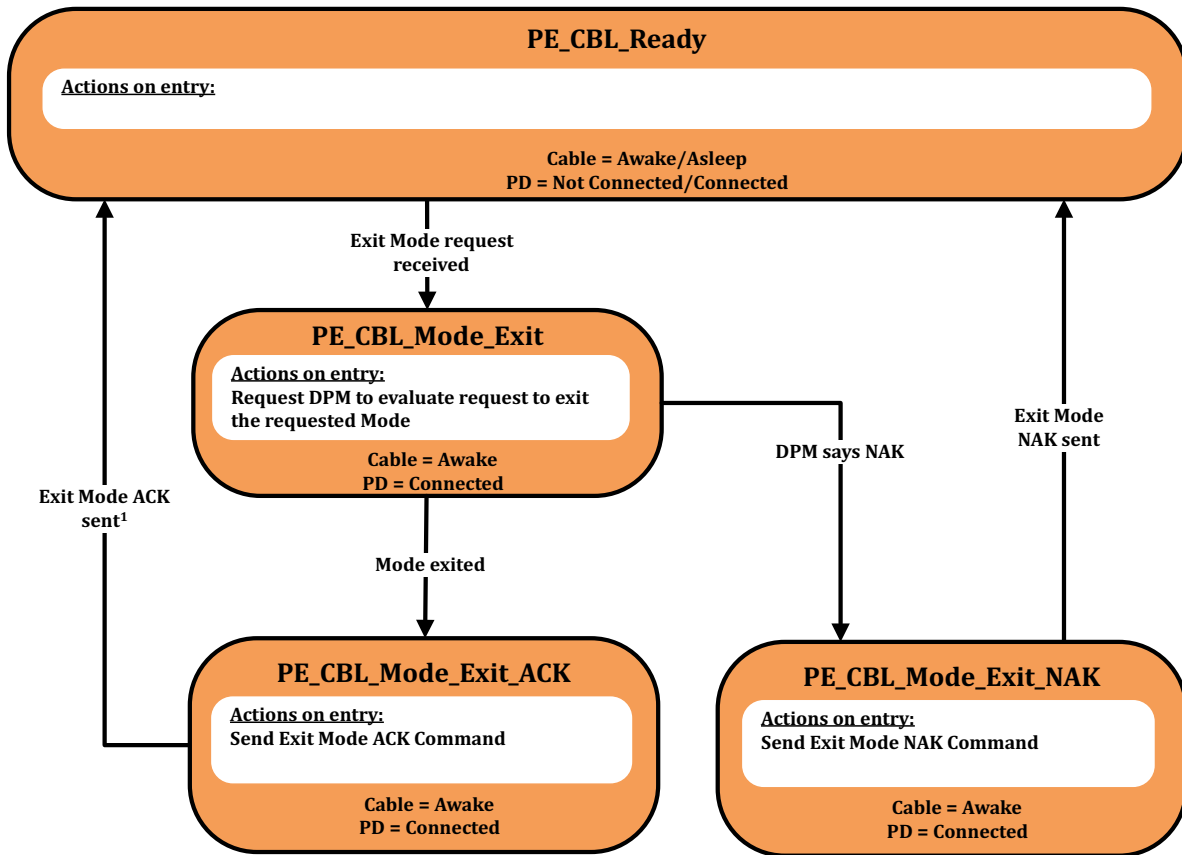
On entry to the *PE_CBL_Mode_Entry_NAK* state the *Policy Engine* **Shall** send a *Structured VDM Enter Mode NAK* Command response as indicated by the *Device Policy Manager*.

The *Policy Engine* **Shall** transition to the *PE_CBL_Ready* state when:

- The *Structured VDM Enter Mode NAK* Command response has been sent.

Figure 8.210, "Cable Plug Structured VDM Exit Mode State Diagram" shows the state diagram for a *Cable Plug* in response to an *Exit Mode* Command.

Figure 8.210 Cable Plug Structured VDM Exit Mode State Diagram



¹⁾ The Cable is required to be in USB operation or USB Safe State at this point.

8.3.3.25.4.2.1

PE_CBL_Mode_Exit State

The *Policy Engine* transitions to the *PE_CBL_Mode_Exit* state from the *PE_CBL_Ready* state when:

- A Structured VDM *Exit Mode* Command request is received from the *DFP*.

On entry to the *PE_CBL_Mode_Exit* state the *Policy Engine* **Shall** request the *Device Policy Manager* to exit the Mode indicated in the *Command*.

The *Policy Engine* **Shall** transition to the *PE_CBL_Mode_Exit_ACK* state when:

- The *Device Policy Manager* indicates that the Mode has been exited.

The *Policy Engine* **Shall** transition to the *PE_CBL_Mode_Exit_NAK* state when:

- The *Device Policy Manager* indicates that the *Command* response to the *Exit Mode* Command request is *NAK*.

8.3.3.25.4.2.2

PE_CBL_Mode_Exit_ACK State

On entry to the *PE_CBL_Mode_Exit_ACK* state the *Policy Engine* **Shall** send a Structured VDM *Exit Mode ACK* Command response.

The *Policy Engine* **Shall** transition to the *PE_CBL_Ready* state when:

- The *Structured VDM Exit Mode ACK* Command response has been sent.

8.3.3.25.4.2.3 PE_CBL_Mode_Exit_NAK State

On entry to the *PE_CBL_Mode_Exit_NAK* state the *Policy Engine* **Shall** send a *Structured VDM Exit Mode NAK* Command response as indicated by the *Device Policy Manager*.

The *Policy Engine* **Shall** transition to the *PE_CBL_Ready* state when:

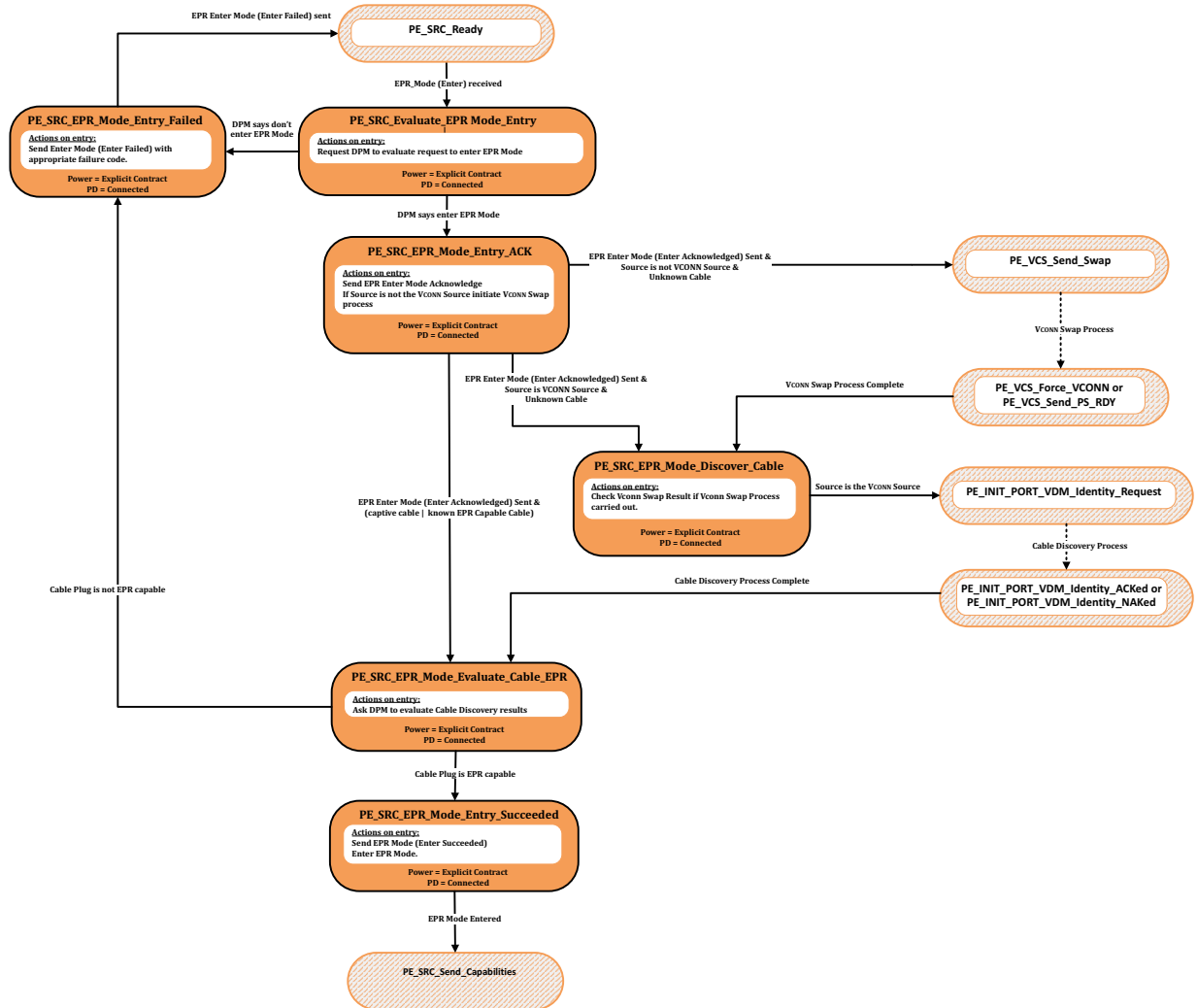
- The *Structured VDM Exit Mode NAK* Command response has been sent.

8.3.3.26 EPR Mode State Diagrams

8.3.3.26.1 Source EPR Mode Entry State Diagram

Figure 8.211, "Source EPR Mode Entry State Diagram" shows the state diagram for an *EPR Source* in response to an *EPR_Mode* Message.

Figure 8.211 Source EPR Mode Entry State Diagram



8.3.3.26.1.1 PE_SRC_Evaluate_EPR_Mode_Entry State

The *Policy Engine* transitions to the *PE_SRC_Evaluate_EPR_Mode_Entry* state from the *PE_SRC_Ready* state when:

- An *EPR_Mode* (Enter) Message is received from the *Sink*.

On Entry to the *PE_SRC_Evaluate_EPR_Mode_Entry* state the *Policy Engine* **shall** request the *Device Policy Manager* to evaluate the *EPR_Mode* (Enter) Message.

The *Policy Engine* **shall** transition to the *PE_SRC_EPR_Mode_Entry_Ack* state when:

- The *Device Policy Manager* indicates that *EPR Mode* can be entered.

The *Policy Engine* **shall** transition to the *PE_SRC_EPR_Mode_Entry_Failed* state when:

- The *Device Policy Manager* indicates that the *EPR Mode* is not to be entered.

8.3.3.26.1.2 PE_SRC_EPR_Mode_Entry_Ack State

On entry to the **PE_SRC_EPR_Mode_Entry_Ack** state the *Policy Engine* **Shall** send a **EPR_Mode** (Enter Acknowledged) *Message*.

The *Policy Engine* **Shall** transition to the **PE_SRC_EPR_Mode_Evaluate_Cable_EPR** state when:

- The **EPR_Mode** (Enter Acknowledged) *Message* has been sent and
- The *Source* is not the *VCONN Source* and
- The cable is a captive cable or a known *EPR Cable*.

The *Policy Engine* **Shall** transition to the **PE_VCS_Send_Swap** state when:

- The **EPR_Mode** (Enter Acknowledged) *Message* has been sent and
- The *Source* is not the *VCONN Source* and
- The cable is unknown.

The *Policy Engine* **Shall** transition to the **PE_SRC_EPR_Mode_Discover_Cable** state when:

- The **EPR_Mode** (Enter Acknowledged) *Message* has been sent and
- The *Source* is the *VCONN Source* and
- The cable is unknown.

8.3.3.26.1.3 PE_SRC_EPR_Mode_Discover_Cable State

The *Policy Engine* transitions to the **PE_SRC_EPR_Mode_Discover_Cable** state from the **PE_VCS_Force_VCONN** state or **PE_VCS_Send_Ps_Rdy** state when:

- A *Source* initiated *VCONN Swap* process has completed.

The *Policy Engine* **Shall** transition to the **PE_INIT_PORT_VDM_Identity_Request** state in order to perform *Cable Plug* discovery when:

- The *Source* is the *VCONN Source*.

The *Policy Engine* **Shall** transition to the **PE_SRC_EPR_Mode_Entry_Failed** state when:

- The *VCONN Swap* process failed (the *Source* is not the *VCONN Source*).

8.3.3.26.1.4 PE_SRC_EPR_Mode_Evaluate_Cable_EPR State

In the **PE_SRC_EPR_Mode_Evaluate_Cable_EPR** state the *Policy Engine* requests the DPM to evaluate the *Cable Discovery* results.

The *Policy Engine* **Shall** transition to the **PE_SRC_EPR_Mode_Entry_Succeeded** state when:

- The *Cable Plug* is capable of *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_EPR_Mode_Entry_Failed** state when:

- The *Cable Plug* is not capable of *EPR Mode*.

8.3.3.26.1.5 PE_SRC_EPR_Mode_Entry_Succeeded State

On entry to the **PE_SRC_EPR_Mode_Entry_Succeeded** state the *Policy Engine* **Shall** send a **EPR_Mode** (Enter Succeeded) *Message* and enter *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Send_Capabilities** state when:

- *EPR Mode* has been entered.

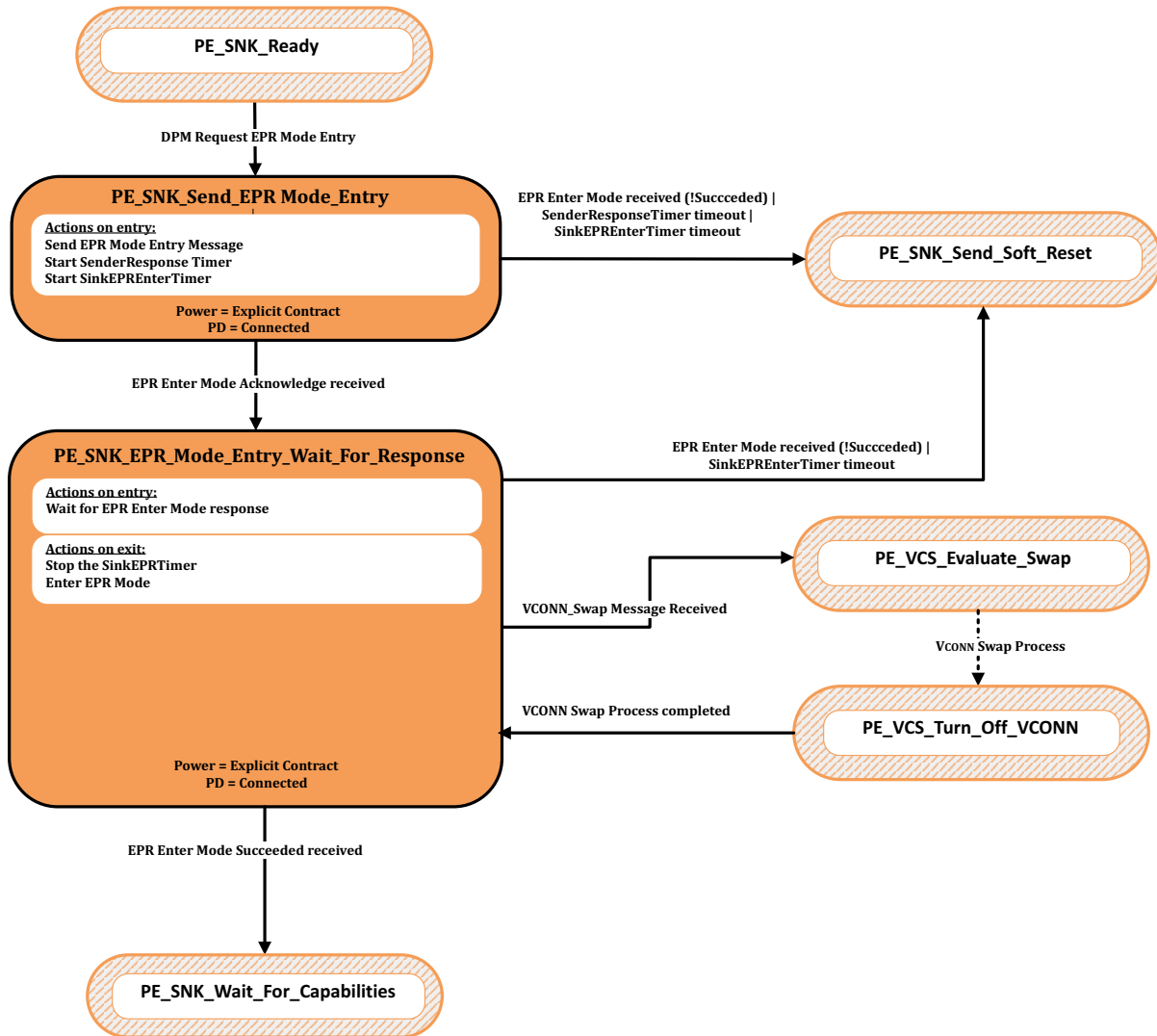
On entry to the **PE_SRC_EPR_Mode_Entry_Failed** state the *Policy Engine* **Shall** send a **EPR_Mode** (Enter Failed) *Message*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Ready** state when:

- The **EPR_Mode** (Enter Failed) *Message* has been sent.

Figure 8.212, "Sink EPR Mode Entry State Diagram" shows the state diagram for an *EPR Sink* initiating the *EPR Mode* Entry process.

Figure 8.212 Sink EPR Mode Entry State Diagram



8.3.3.26.2.1

PE_SNK_Send_EPR_Mode_Entry State

The *Policy Engine* transitions to the *PE_SNK_Send_EPR_Mode_Entry* state from the *PE_SNK_Ready* state when:

- The DPM requests entry into *EPR Mode*.

On Entry to the *PE_SNK_Send_EPR_Mode_Entry* state the *Policy Engine* **Shall** send an *EPR_Mode* (Enter) *Message* and starts the *SenderResponseTimer* and the *SinkEPREnterTimer*.

Note: The *SinkEPREnterTimer* **Shall** continue to run in every state until it is stopped or times out.

The *Policy Engine* **Shall** transition to the *PE_SNK_EPR_Mode_Wait_For_Response* state when:

- An *EPR_Mode* (Enter Acknowledge) *Message* is received.

The *Policy Engine* **Shall** transition to the *PE_SNK_Send_Soft_Reset* state when:

- An *EPR_Mode* *Message* is received which is not Enter Succeeded or

- The *SenderResponseTimer* times out or
- The *SinkEPREnterTimer* times out.

8.3.3.26.2.2 PE_SNK_EPR_Mode_Wait_For_Response State

In the State the *Policy Engine* waits for a confirmation that the *EPR Mode* entry request has succeeded.

On exit from the *PE_SNK_EPR_Mode_Wait_For_Response* state the *Policy Engine* **Shall** stop the *SinkEPREnterTimer* and enter *EPR Mode*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Send_Soft_Reset* state when:

- An *EPR_Mode* Message is received which is not Enter Succeeded or
- The *SinkEPREnterTimer* times out.

The *Policy Engine* **Shall** transition to the *PE_VCS_Evaluate_Swap* State when:

- A *VCONN_Swap* Message is received.

The *Policy Engine* **Shall** transition back from the *PE_VCS_Turn_Off_VCONN* State to the *PE_SNK_EPR_Mode_Wait_For_Response* State when:

- The *VCONN Swap* process has completed.

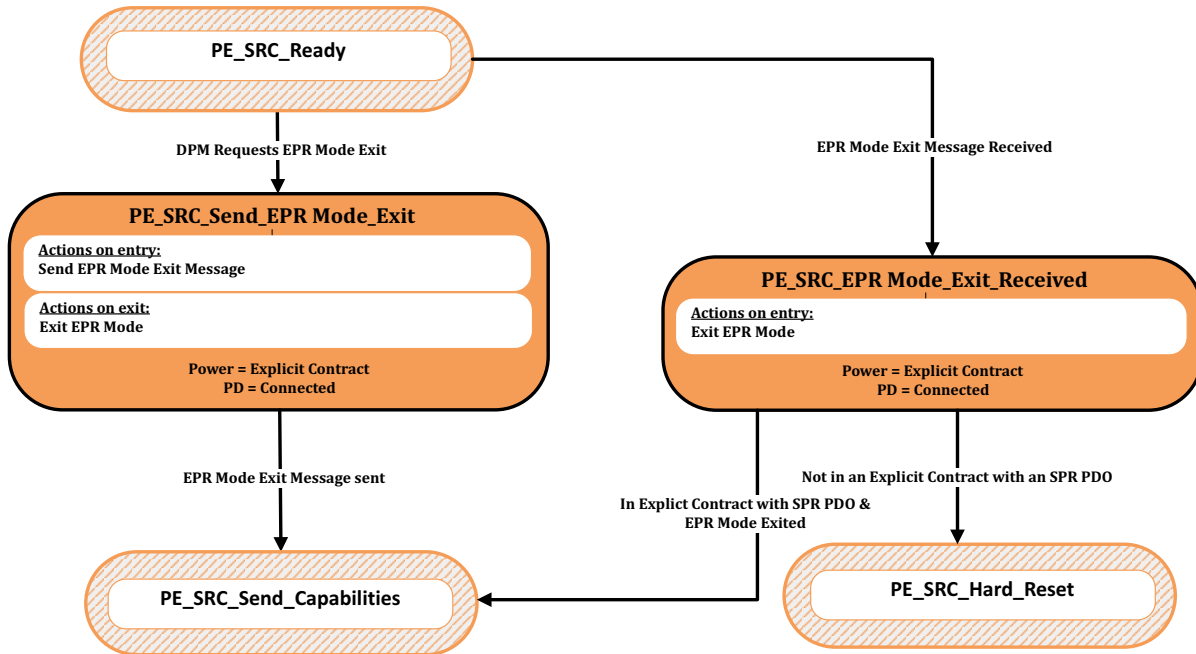
The *Policy Engine* **Shall** transition to the *PE_SNK_Wait_for_Capabilities* state when:

- An *EPR_Mode* (Enter Succeeded) Message has been received.

8.3.3.26.3 Source EPR Mode Exit State Diagram

Figure 8.213, "Source EPR Mode Exit State Diagram" shows the state diagram for an *EPR Source* initiating the *EPR Mode* exit process.

Figure 8.213 Source EPR Mode Exit State Diagram



8.3.3.26.3.1 PE_SRC_Send_EPR_Mode_Exit State

The *Policy Engine* transitions to the **PE_SRC_Send_EPR_Mode_Exit** state from the **PE_SRC_Ready** state when:

- The DPM requests exit from *EPR Mode*.

On Entry to the **PE_SRC_Send_EPR_Mode_Exit** state the *Policy Engine* **Shall** send an **EPR_Mode** (Exit) *Message*.

On Exit from the **PE_SRC_Send_EPR_Mode_Exit** state the *Policy Engine* **Shall** exit *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Send_Capabilities** state when:

- The **EPR_Mode** (Exit) *Message* has been sent.

8.3.3.26.3.2 PE_SRC_EPR_Mode_Exit_Received State

The *Policy Engine* transitions to the **PE_SRC_EPR_Mode_Exit_Received** state from the **PE_SRC_Ready** state when:

- An **EPR_Mode** (Exit) *Message* is received.

On Entry to the **PE_SRC_EPR_Mode_Exit_Received** state the *Policy Engine* **Shall** exit *EPR Mode*.

The *Policy Engine* **Shall** transition to the **PE_SRC_Send_Capabilities** state when:

- In an *Explicit Contract* with an *SPR (A)PDO* and
- *EPR Mode* has been exited.

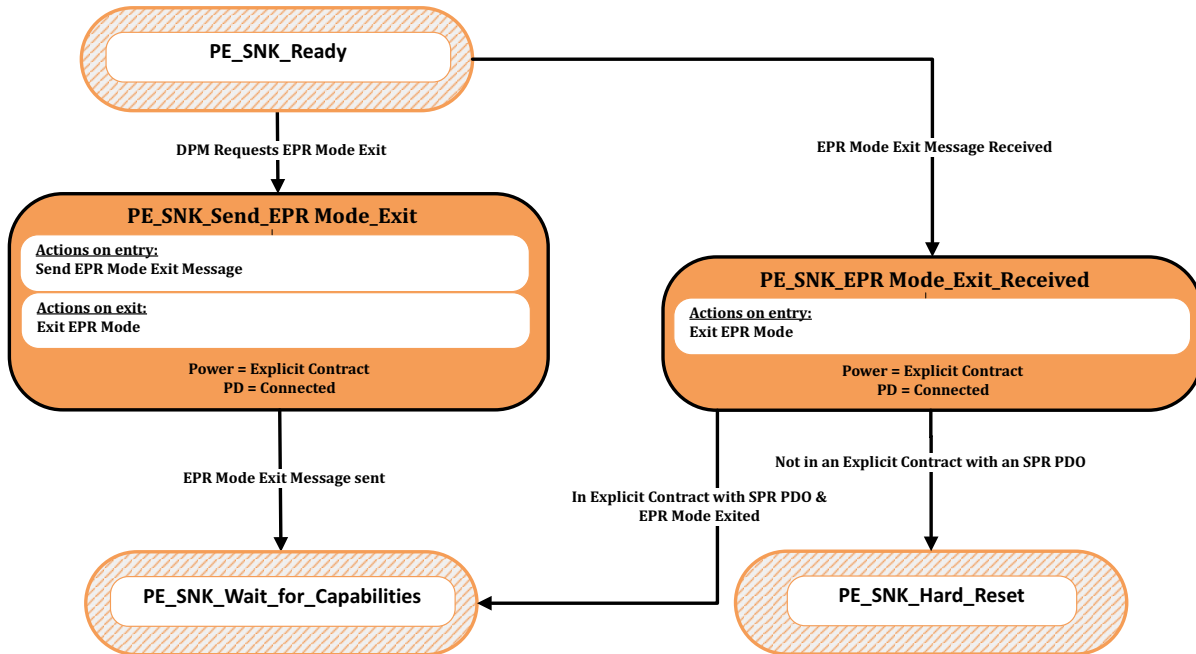
The *Policy Engine* **Shall** transition to the **PE_SRC_Hard_Reset** state when:

- Not in an *Explicit Contract* with an *SPR (A)PDO*.

8.3.3.26.4 Sink EPR Mode Exit State Diagram

Figure 8.214, "Sink EPR Mode Exit State Diagram" shows the state diagram for an *EPR Sink* initiating the *EPR Mode* exit process.

Figure 8.214 Sink EPR Mode Exit State Diagram



8.3.3.26.4.1 PE_SNK_Send_EPR_Mode_Exit State

The *Policy Engine* transitions to the *PE_SNK_Send_EPR_Mode_Exit* state from the *PE_SNK_Ready* state when:

- The DPM requests exit from *EPR Mode*.

On Entry to the *PE_SNK_Send_EPR_Mode_Exit* state the *Policy Engine* **Shall** send an *EPR_Mode* (Exit) *Message*.

On Exit from the *PE_SNK_Send_EPR_Mode_Exit* state the *Policy Engine* **Shall** exit *EPR Mode*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Wait_for_Capabilities* state when:

- The *EPR_Mode* (Exit) *Message* has been sent.

8.3.3.26.4.2 PE_SNK_EPR_Mode_Exit_Received State

The *Policy Engine* transitions to the *PE_SNK_EPR_Mode_Exit_Received* state from the *PE_SNK_Ready* state when:

- An *EPR_Mode* (Exit) *Message* is received.

On Entry to the *PE_SNK_EPR_Mode_Exit_Received* state the *Policy Engine* **Shall** exit *EPR Mode*.

The *Policy Engine* **Shall** transition to the *PE_SNK_Wait_for_Capabilities* state when:

- In an *Explicit Contract* with an *SPR (A)PDO* and
- EPR Mode* has been exited.

The *Policy Engine* **Shall** transition to the *PE_SNK_Hard_Reset* state when:

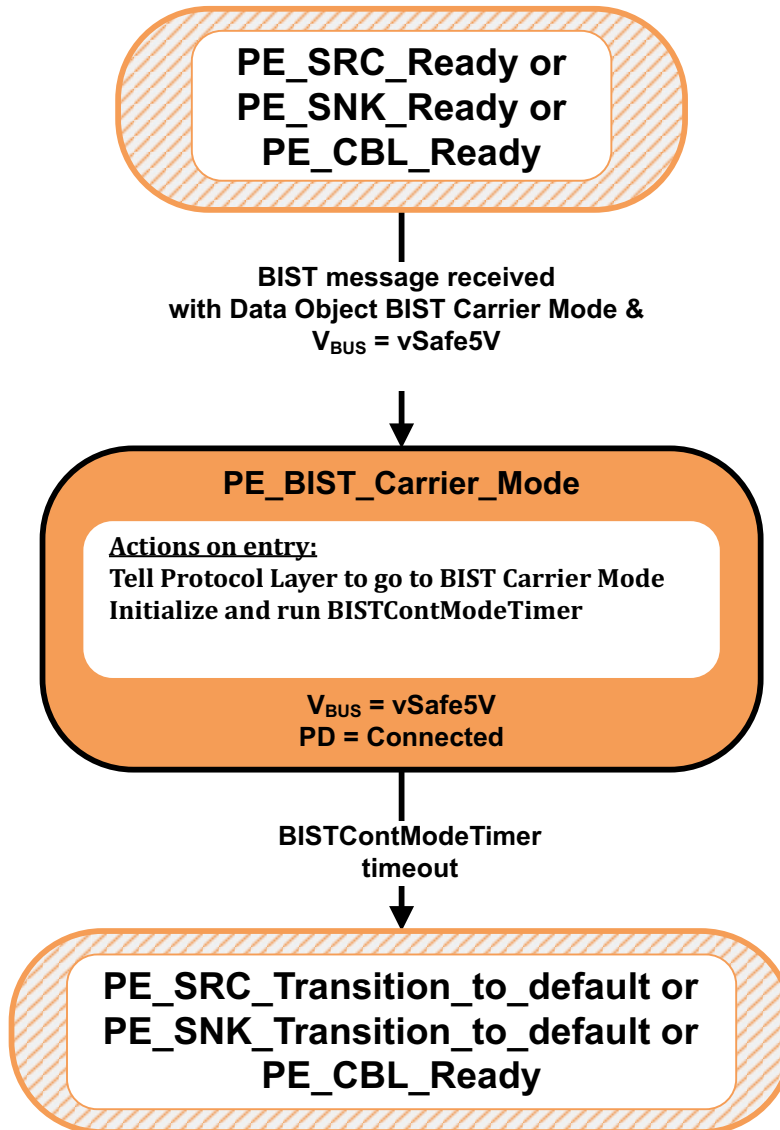
- Not in an *Explicit Contract* with an *SPR (A)PDO*.

8.3.3.27 BIST State diagrams

8.3.3.27.1 BIST Carrier Mode State Diagram

[Figure 8.215, "BIST Carrier Mode State Diagram"](#) shows the state diagram required by a *UUT*, which can be either a *Source*, *Sink* or *Cable Plug*, when operating in *BIST Carrier Mode*. Transitions **shall** be from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* states.

Figure 8.215 BIST Carrier Mode State Diagram



8.3.3.27.1.1 PE_BIST_Carrier_Mode State

The *Source*, *Sink* or *Cable Plug* **shall** enter the *PE_BIST_Carrier_Mode* state from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- A **BIST** Message is received with a **BIST Carrier Mode** BIST Data Object and
- V_{BUS} is at **vSafe5V**.

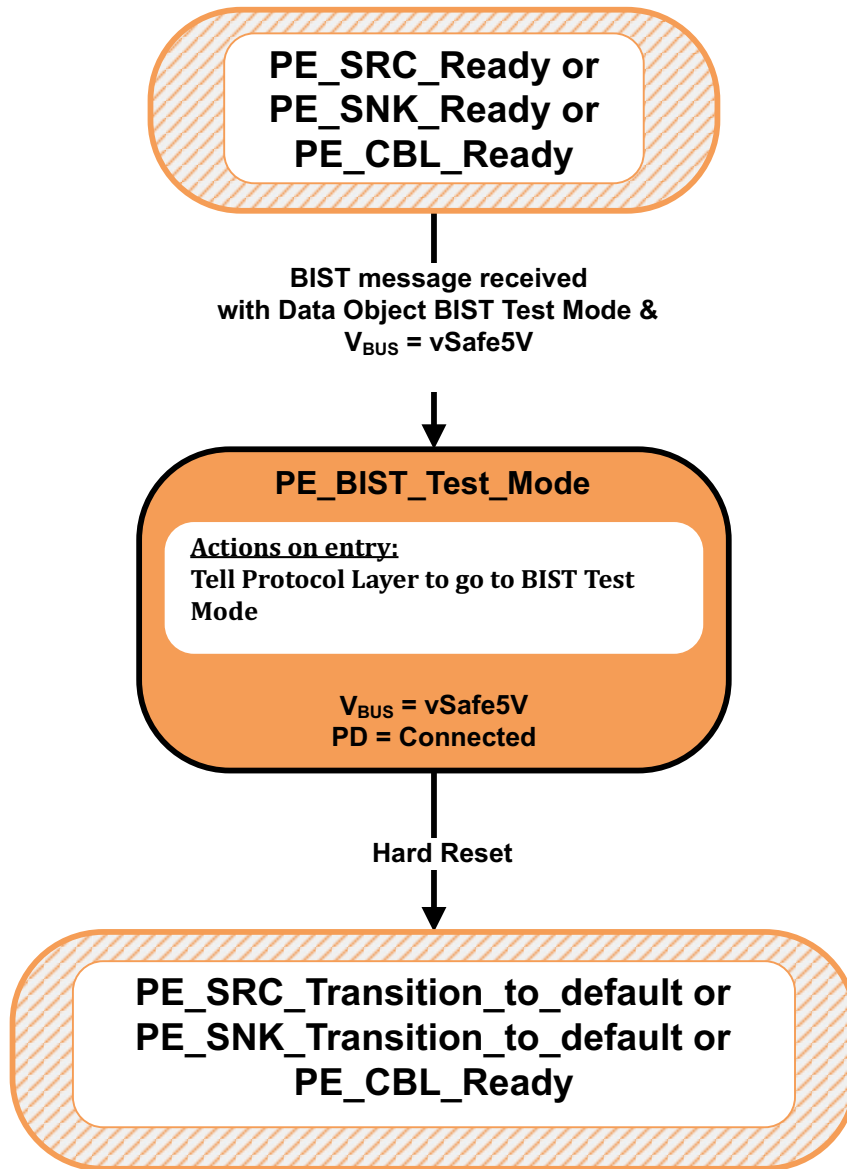
On entry to the *PE_BIST_Carrier_Mode* state the Policy Engine **shall** tell the Protocol Layer to go to BIST Carrier Mode (see [Section 6.4.3.1, "BIST Carrier Mode"](#)) and **shall** initialize and run the **BISTContModeTimer**.

The *Policy Engine* **Shall** transition to either the *PE_SRC_Transition_to_default* state, *PE_SNK_Transition_to_default* state or *PE_CBL_Ready* state (as appropriate) when:

- The *BISTContModeTimer* times out.

Figure 8.216, "BIST Test Data Mode State Diagram" shows the state diagram required by a *UUT*, which can be either a *Source*, *Sink* or *Cable Plug*, when operating in *BIST Test Data Mode*. Transitions **shall** be from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* states.

Figure 8.216 BIST Test Data Mode State Diagram



8.3.3.27.2.1

PE_BIST_Test_Mode State

The *Source*, *Sink* or *Cable Plug* **shall** enter the *PE_BIST_Test_Mode* state from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- A **BIST** Message is received with a **BIST Test Data** BIST Data Object and
- V_{BUS} is at **vSafe5V**.

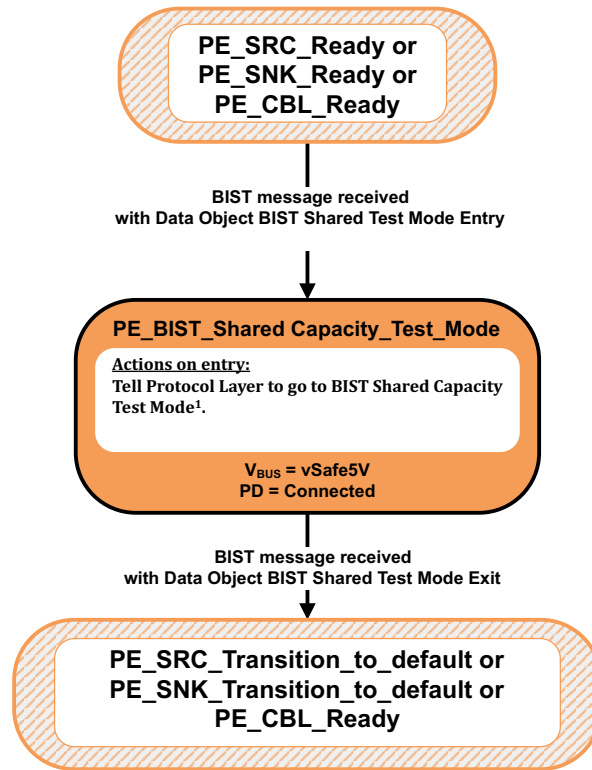
On entry to the **PE_BIST_Test_Mode** state the *Policy Engine* **Shall** tell the *Protocol Layer* to go into *BIST Test Data Mode* where it sends no further *Messages* except for **GoodCRC** *Messages* in response to received *Messages* (see [Section 6.4.3.2, "BIST Test Data Mode"](#)).

The *Policy Engine* **Shall** transition to either the **PE_SRC_Transition_to_default** state, **PE_SNK_Transition_to_default** state or **PE_CBL_Ready** state (as appropriate) when:

- A *Hard Reset* occurs.

Figure 8.217, "BIST Shared Capacity Test Mode State Diagram" shows the state diagram required by a UUT, which can be either a *Source*, *Sink* or *Cable Plug*, when operating in *BIST Shared Capacity Test Mode*. Transitions **shall** be from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* states.

Figure 8.217 BIST Shared Capacity Test Mode State Diagram



- ¹⁾ The UUT **shall** exit BIST Shared Capacity Test Mode when It is powered off. The UUT **shall** remain in BIST Shared Capacity Test Mode for any PD event (except when a *BIST Shared Test Mode Exit* BIST Data Object, is received); specifically the UUT **shall** remain in BIST Shared Capacity Test Mode when any of the following PD events occurs: Hard Reset, Cable Reset, Soft Reset, Data Role Swap, Power Role Swap, Fast Role Swap, VCONN Swap. The UUT **may** leave test mode if the tester makes a request that exceeds the capabilities of the UUT.

8.3.3.27.3.1

PE_BIST_Shared_Capacity_Test_Mode State

The *Source*, *Sink* or *Cable Plug* **shall** enter the *PE_BIST_Shared_Capacity_Test_Mode* state from either the *PE_SRC_Ready*, *PE_SNK_Ready* or *PE_CBL_Ready* state when:

- A *BIST* Message is received with a *BIST Shared Test Mode Entry* BIST Data Object and
- *V_{BUS}* is at *vSafe5V*.

On entry to the *PE_BIST_Shared_Capacity_Test_Mode* state the Policy Engine **shall** tell the Protocol Layer to go to BIST Shared Capacity Test Mode (see [Section 6.4.3.3, "BIST Shared Capacity Test Mode"](#)).

The Policy Engine **shall** transition to either the *PE_SRC_Transition_to_default* state, *PE_SNK_Transition_to_default* state or *PE_CBL_Ready* state (as appropriate) when:

- A *BIST* Message is received with a *BIST Shared Test Mode Exit* BIST Data Object.

8.3.3.28 USB Type-C Referenced States

This section contains states cross-referenced from the [\[USB Type-C 2.4\]](#) specification.

8.3.3.28.1 ErrorRecovery state

The **ErrorRecovery** state is used to electronically disconnect *Port Partners* using the *USB Type-C* connector. The **ErrorRecovery** state **Shall** be entered when there are errors on *USB Type-C* Ports which cannot be recovered by *Hard Reset*. The **ErrorRecovery** state **Shall** map to *USB Type-C ErrorRecovery* state operation as defined in the [\[USB Type-C 2.4\]](#) specification. Bus powered Sinks **Shall Not** be required to meet this requirement as removal of their power will serve the same purpose.

On entry to the **ErrorRecovery** state the *Explicit Contract* and PD Connection **Shall** be ended.

On exit from the **ErrorRecovery** state a new *Explicit Contract* **Should** be established once the *Port Partners* have re-connected over the *CC* wire.

8.3.3.29 Policy Engine States

Table 8.154, "Policy Engine States" lists the states used by the various state machines.

Table 8.154 Policy Engine States

State name	Reference
SenderResponseTimer	
<i>SRT_Stopped</i>	Section 8.3.3.1.1.1
<i>SRT_Running</i>	Section 8.3.3.1.1.2
<i>SRT_Expired</i>	Section 8.3.3.1.1.3
Source Port	
<i>PE_SRC_Startup</i>	Section 8.3.3.2.1
<i>PE_SRC_Discovery</i>	Section 8.3.3.2.2
<i>PE_SRC_Send_Capabilities</i>	Section 8.3.3.2.3
<i>PE_SRC_Negotiate_Capability</i>	Section 8.3.3.2.4
<i>PE_SRC_Transition_Supply</i>	Section 8.3.3.2.5
<i>PE_SRC_Ready</i>	Section 8.3.3.2.6
<i>PE_SRC_Disabled</i>	Section 8.3.3.2.7
<i>PE_SRC_Capability_Response</i>	Section 8.3.3.2.8
<i>PE_SRC_Hard_Reset</i>	Section 8.3.3.2.9
<i>PE_SRC_Hard_Reset_Received</i>	Section 8.3.3.2.10
<i>PE_SRC_Transition_to_default</i>	Section 8.3.3.2.11
<i>PE_SRC_Give_Source_Cap</i>	Section 8.3.3.2.15
<i>PE_SRC_Get_Sink_Cap</i>	Section 8.3.3.2.12
<i>PE_SRC_Wait_New_Capabilities</i>	Section 8.3.3.2.13
<i>PE_SRC_EPR_Keep_Alive</i>	Section 8.3.3.2.14
Sink Port	
<i>PE_SNK_Startup</i>	Section 8.3.3.3.1
<i>PE_SNK_Discovery</i>	Section 8.3.3.3.2
<i>PE_SNK_Wait_for_Capabilities</i>	Section 8.3.3.3.3
<i>PE_SNK_Evaluate_Capability</i>	Section 8.3.3.3.4
<i>PE_SNK_Select_Capability</i>	Section 8.3.3.3.5
<i>PE_SNK_Transition_Sink</i>	Section 8.3.3.3.6
<i>PE_SNK_Ready</i>	Section 8.3.3.3.7
<i>PE_SNK_Hard_Reset</i>	Section 8.3.3.3.8
<i>PE_SNK_Transition_to_default</i>	Section 8.3.3.3.9
<i>PE_SNK_Give_Sink_Cap</i>	Section 8.3.3.3.10
<i>PE_SNK_Get_Source_Cap</i>	Section 8.3.3.3.12
<i>PE_SNK_EPR_Keep_Alive</i>	Section 8.3.3.3.11

Table 8.154 Policy Engine States

State name	Reference
Soft Reset and Protocol Error	
Source Port Soft Reset	
<i>PE_SRC_Send_Soft_Reset</i>	Section 8.3.3.4.1.1
<i>PE_SRC_Soft_Reset</i>	Section 8.3.3.4.1.2
Sink Port Soft Reset	
<i>PE_SNK_Send_Soft_Reset</i>	Section 8.3.3.4.2.1
<i>PE_SNK_Soft_Reset</i>	Section 8.3.3.4.2.2
Data Reset	
DFP Data Reset	
<i>PE_DDR_Send_Data_Reset</i>	Section 8.3.3.5.1.1
<i>PE_DDR_Data_Reset_Received</i>	Section 8.3.3.5.1.2
<i>PE_DDR_Wait_For_VCONN_Off</i>	Section 8.3.3.5.1.3
<i>PE_DDR_Perform_Data_Reset</i>	Section 8.3.3.5.1.4
UFP Data Reset	
<i>PE_UDR_Send_Data_Reset</i>	Section 8.3.3.5.2.1
<i>PE_UDR_Data_Reset_Received</i>	Section 8.3.3.5.2.2
<i>PE_UDR_Turn_Off_VCONN</i>	Section 8.3.3.5.2.3
<i>PE_UDR_Send_Ps_Rdy</i>	Section 8.3.3.5.2.4
<i>PE_UDR_Wait_For_Data_Reset_Complete</i>	Section 8.3.3.5.2.5
Not Supported Message	
Source Port Not Supported	
<i>PE_SRC_Send_Not_Supported</i>	Section 8.3.3.6.1.1
<i>PE_SRC_Not_Supported_Received</i>	Section 8.3.3.6.1.2
<i>PE_SRC_Chunk_Received</i>	Section 8.3.3.6.1.3
Sink Port Not Supported	
<i>PE_SNK_Send_Not_Supported</i>	Section 8.3.3.6.2.1
<i>PE_SNK_Not_Supported_Received</i>	Section 8.3.3.6.2.2
<i>PE_SNK_Chunk_Received</i>	Section 8.3.3.6.2.3
Source Alert	
Source Port Source Alert	
<i>PE_SRC_Send_Source_Alert</i>	Section 8.3.3.7.1.1
<i>PE_SRC_Wait_for_Get_Status</i>	Section 8.3.3.7.1.2
Sink Port Source Alert	
<i>PE_SNK_Source_Alert_Received</i>	Section 8.3.3.7.2.1
Sink Port Sink Alert	
<i>PE_SNK_Send_Sink_Alert</i>	Section 8.3.3.7.3.1
<i>PE_SNK_Wait_for_Get_Status</i>	Section 8.3.3.7.3.2
Source Port Sink Alert	
<i>PE_SRC_Sink_Alert_Received</i>	Section 8.3.3.7.4.1

Table 8.154 Policy Engine States

State name	Reference
Source/Sink Extended Capabilities	
Sink Port Get Source Capabilities Extended	
<i>PE_SNK_Get_Source_Cap_Ext</i>	Section 8.3.3.8.1.1
Source Port Give Source Capabilities Extended	
<i>PE_SRC_Give_Source_Cap_Ext</i>	Section 8.3.3.8.2.1
Source Port Get Sink Capabilities Extended	
<i>PE_SRC_Get_Sink_Cap_Ext</i>	Section 8.3.3.8.3.1
Source Port Give Sink Capabilities Extended	
<i>PE_SNK_Give_Sink_Cap_Ext</i>	Section 8.3.3.8.4.1
Source Information	
Sink Port Get Source Information	
<i>PE_SNK_Get_Source_Info</i>	Section 8.3.3.9.1.1
Source Port Give Source Information	
<i>PE_SRC_Give_Source_Info</i>	Section 8.3.3.9.2.1
Status	
Get Status	
<i>PE_Get_Status</i>	Section 8.3.3.10.1.1
Give Status	
<i>PE_Give_Status</i>	Section 8.3.3.10.1.1
Sink Port Get PPS Status	
<i>PE_SNK_Get_PPS_Status</i>	Section 8.3.3.10.3.1
Source Port Give PPS Status	
<i>PE_SRC_Give_PPS_Status</i>	Section 8.3.3.10.4.1
Battery Capabilities	
Get Battery Capabilities	
<i>PE_Get_Battery_Cap</i>	Section 8.3.3.11.1.1
Give Battery Capabilities	
<i>PE_Give_Battery_Cap</i>	Section 8.3.3.11.2.1
Battery Status	
Get Battery Status	
<i>PE_Get_Battery_Status</i>	Section 8.3.3.12.1.1
Give Battery Status	
<i>PE_Give_Battery_Status</i>	Section 8.3.3.12.2.1
Manufacturer Information	
Get Manufacturer Information	
<i>PE_Get_Manufacturer_Info</i>	Section 8.3.3.13.1.1
Give Manufacturer Information	
<i>PE_Give_Manufacturer_Info</i>	Section 8.3.3.13.2.1
Country Codes and Information	
Get Country Codes	
<i>PE_Get_Country_Codes</i>	Section 8.3.3.14.1.1

Table 8.154 Policy Engine States

State name	Reference
Give Country Codes	
<i>PE_Give_Country_Codes</i>	Section 8.3.3.14.2.1
Get Country Information	
<i>PE_Get_Country_Info</i>	Section 8.3.3.14.3.1
Give Country Information	
<i>PE_Give_Country_Info</i>	Section 8.3.3.14.4.1
Revision	
Get Revision	
<i>PE_Get_Revision</i>	Section 8.3.3.15.1.1
Give Revision	
<i>PE_Give_Revision</i>	Section 8.3.3.15.2.1
Enter USB	
DFP Enter USB	
<i>PE_DEU_Send_Enter_USB</i>	Section 8.3.3.16.1.1
UFP Enter USB	
<i>PE_UEU_Enter_USB_Received</i>	Section 8.3.3.16.2.1
Security Request/Response	
Send Security Request	
<i>PE_Send_Security_Request</i>	Section 8.3.3.17.1.1
Send Security Response	
<i>PE_Send_Security_Response</i>	Section 8.3.3.17.2.1
Security Response Received	
<i>PE_Security_Response_Received</i>	Section 8.3.3.17.3.1
Firmware Update Request/Response	
Send Firmware Update Request	
<i>PE_Send_Firmware_Update_Request</i>	Section 8.3.3.18.1.1
Send Firmware Update Response	
<i>PE_Send_Firmware_Update_Response</i>	Section 8.3.3.18.2.1
Firmware Update Response Received	
<i>PE_Firmware_Update_Response_Received</i>	Section 8.3.3.18.3.1
Dual-Role Port	
DFP to UFP Data Role Swap	
<i>PE_DRS_DFP_UFP_Evaluate_Swap</i>	Section 8.3.3.19.1.2
<i>PE_DRS_DFP_UFP_Accept_Swap</i>	Section 8.3.3.19.1.3
<i>PE_DRS_DFP_UFP_Change_to_UFP</i>	Section 8.3.3.19.1.4
<i>PE_DRS_DFP_UFP_Send_Swap</i>	Section 8.3.3.19.1.5
<i>PE_DRS_DFP_UFP_Reject_Swap</i>	Section 8.3.3.19.1.6

Table 8.154 Policy Engine States

State name	Reference
UFP to DFP Data Role Swap	
<i>PE_DRS_UFP_DFP_Evaluate_Swap</i>	Section 8.3.3.19.2.2
<i>PE_DRS_UFP_DFP_Accept_Swap</i>	Section 8.3.3.19.2.3
<i>PE_DRS_UFP_DFP_Change_to_DFP</i>	Section 8.3.3.19.2.4
<i>PE_DRS_UFP_DFP_Send_Swap</i>	Section 8.3.3.19.2.5
<i>PE_DRS_UFP_DFP_Reject_Swap</i>	Section 8.3.3.19.2.6
Source to Sink Power Role Swap	
<i>PE_PRS_SRC_SNK_Evaluate_Swap</i>	Section 8.3.3.19.3.2
<i>PE_PRS_SRC_SNK_Accept_Swap</i>	Section 8.3.3.19.3.3
<i>PE_PRS_SRC_SNK_Transition_to_off</i>	Section 8.3.3.19.3.4
<i>PE_PRS_SRC_SNK_Assert_Rd</i>	Section 8.3.3.19.3.5
<i>PE_PRS_SRC_SNK_Wait_Source_on</i>	Section 8.3.3.19.3.6
<i>PE_PRS_SRC_SNK_Send_Swap</i>	Section 8.3.3.19.3.7
<i>PE_PRS_SRC_SNK_Reject_Swap</i>	Section 8.3.3.19.3.8
Sink to Source Power Role Swap	
<i>PE_PRS_SNK_SRC_Evaluate_Swap</i>	Section 8.3.3.19.4.2
<i>PE_PRS_SNK_SRC_Accept_Swap</i>	Section 8.3.3.19.4.3
<i>PE_PRS_SNK_SRC_Transition_to_off</i>	Section 8.3.3.19.4.4
<i>PE_PRS_SNK_SRC_Assert_Rp</i>	Section 8.3.3.19.4.5
<i>PE_PRS_SNK_SRC_Source_on</i>	Section 8.3.3.19.4.6
<i>PE_PRS_SNK_SRC_Send_Swap</i>	Section 8.3.3.19.4.7
<i>PE_PRS_SNK_SRC_Reject_Swap</i>	Section 8.3.3.19.4.8
Source to Sink Fast Role Swap	
<i>PE_FRS_SRC_SNK_Evaluate_Swap</i>	Section 8.3.3.19.5.2
<i>PE_FRS_SRC_SNK_Accept_Swap</i>	Section 8.3.3.19.5.3
<i>PE_FRS_SRC_SNK_Transition_to_off</i>	Section 8.3.3.19.5.4
<i>PE_FRS_SRC_SNK_Assert_Rd</i>	Section 8.3.3.19.5.5
<i>PE_FRS_SRC_SNK_Wait_Source_on</i>	Section 8.3.3.19.5.6
Sink to Source Fast Role Swap	
<i>PE_FRS_SNK_SRC_Start_AMS</i>	Section 8.3.3.19.6.1
<i>PE_FRS_SNK_SRC_Send_Swap</i>	Section 8.3.3.19.6.2
<i>PE_FRS_SNK_SRC_Transition_to_off</i>	Section 8.3.3.19.6.3
<i>PE_FRS_SNK_SRC_VBUS_Applied</i>	Section 8.3.3.19.6.4
<i>PE_FRS_SNK_SRC_Assert_Rp</i>	Section 8.3.3.19.6.5
<i>PE_FRS_SNK_SRC_Source_on</i>	Section 8.3.3.19.6.6
Dual-Role Source Port Get Source Capabilities	
<i>PE_DR_SRC_Get_Source_Cap</i>	Section 8.3.3.19.7.1
Dual-Role Source Port Give Sink Capabilities	
<i>PE_DR_SRC_Give_Sink_Cap</i>	Section 8.3.3.19.8.1

Table 8.154 Policy Engine States

State name	Reference
Dual-Role Sink Port Get Sink Capabilities	
<i>PE_DR_SNK_Get_Sink_Cap</i>	Section 8.3.3.19.9.1
Dual-Role Sink Port Give Source Capabilities	
<i>PE_DR_SNK_Give_Source_Cap</i>	Section 8.3.3.19.10.1
Dual-Role Source Port Get Source Capabilities Extended	
<i>PE_DR_SRC_Get_Source_Cap_Ext</i>	Section 8.3.3.19.11.1
Dual-Role Sink Port Give Source Capabilities Extended	
<i>PE_DR_SNK_Give_Source_Cap_Ext</i>	Section 8.3.3.19.12.1
Dual-Role Sink Port Get Sink Capabilities Extended	
<i>PE_DR_SNK_Get_Sink_Cap_Ext</i>	Section 8.3.3.19.13.1
Dual-Role Source Port Give Sink Capabilities Extended	
<i>PE_DR_SRC_Give_Sink_Cap_Ext</i>	Section 8.3.3.19.14.1
Dual-Role Source Port Get Source Information	
<i>PE_DR_SRC_Get_Source_Info</i>	Section 8.3.3.19.15.1
Dual-Role Sink Port Give Source Information	
<i>PE_DR_SNK_Give_Source_Info</i>	Section 8.3.3.19.16.1
USB Type-C VCONN Swap	
<i>PE_VCS_Send_Swap</i>	Section 8.3.3.20.1
<i>PE_VCS_Evaluate_Swap</i>	Section 8.3.3.20.2
<i>PE_VCS_Accept_Swap</i>	Section 8.3.3.20.3
<i>PE_VCS_Reject_Swap</i>	Section 8.3.3.20.4
<i>PE_VCS_Wait_For_VCONN</i>	Section 8.3.3.20.5
<i>PE_VCS_Turn_Off_VCONN</i>	Section 8.3.3.20.6
<i>PE_VCS_Turn_On_VCONN</i>	Section 8.3.3.20.7
<i>PE_VCS_Send_Ps_Rdy</i>	Section 8.3.3.20.8
<i>PE_VCS_Force_VCONN</i>	Section 8.3.3.20.9
Initiator Structured VDM	
Initiator to Port Structured VDM Discover Identity	
<i>PE_INIT_PORT_VDM_Identity_Request</i>	Section 8.3.3.21.1.1
<i>PE_INIT_PORT_VDM_Identity_ACKed</i>	Section 8.3.3.21.1.2
<i>PE_INIT_PORT_VDM_Identity_NAKed</i>	Section 8.3.3.21.1.3
Initiator Structured VDM Discover SVIDs	
<i>PE_INIT_VDM_SVIDs_Request</i>	Section 8.3.3.21.2.1
<i>PE_INIT_VDM_SVIDs_ACKed</i>	Section 8.3.3.21.2.2
<i>PE_INIT_VDM_SVIDs_NAKed</i>	Section 8.3.3.21.2.3
Initiator Structured VDM Discover Modes	
<i>PE_INIT_VDM_Modes_Request</i>	Section 8.3.3.21.3.1
<i>PE_INIT_VDM_Modes_ACKed</i>	Section 8.3.3.21.3.2
<i>PE_INIT_VDM_Modes_NAKed</i>	Section 8.3.3.21.3.3

Table 8.154 Policy Engine States

State name	Reference
Initiator Structured VDM Attention	
<i>PE_INIT_VDM_Attention_Request</i>	Section 8.3.3.21.4.1
Responder Structured VDM	
Responder Structured VDM Discovery Identity	
<i>PE_RESP_VDM_Get_Identity</i>	Section 8.3.3.22.1.1
<i>PE_RESP_VDM_Send_Identity</i>	Section 8.3.3.22.1.2
<i>PE_RESP_VDM_Get_Identity_NAK</i>	Section 8.3.3.22.1.3
Responder Structured VDM Discovery SVIDs	
<i>PE_RESP_VDM_Get_SVIDs</i>	Section 8.3.3.22.2.1
<i>PE_RESP_VDM_Send_SVIDs</i>	Section 8.3.3.22.2.2
<i>PE_RESP_VDM_Get_SVIDs_NAK</i>	Section 8.3.3.22.2.3
Responder Structured VDM Discovery Modes	
<i>PE_RESP_VDM_Get_Modes</i>	Section 8.3.3.22.3.1
<i>PE_RESP_VDM_Send_Modes</i>	Section 8.3.3.22.3.2
<i>PE_RESP_VDM_Get_Modes_NAK</i>	Section 8.3.3.22.3.3
Receiving a Structured VDM Attention	
<i>PE_RCV_VDM_Attention_Request</i>	Section 8.3.3.22.4.1
DFP Structured VDM	
DFP Structured VDM Mode Entry	
<i>PE_DFP_VDM_Mode_Entry_Request</i>	Section 8.3.3.23.1.1
<i>PE_DFP_VDM_Mode_Entry_ACKed</i>	Section 8.3.3.23.1.2
<i>PE_DFP_VDM_Mode_Entry_NAKed</i>	Section 8.3.3.23.1.3
DFP Structured VDM Mode Exit	
<i>PE_DFP_VDM_Mode_Exit_Request</i>	Section 8.3.3.23.2.1
<i>PE_DFP_VDM_Mode_Exit_ACKed</i>	Section 8.3.3.23.2.2
UFP Structure VDM	
UFP Structured VDM Enter Mode	
<i>PE_UFP_VDM_Evaluate_Mode_Entry</i>	Section 8.3.3.24.1.1
<i>PE_UFP_VDM_Mode_Entry_ACK</i>	Section 8.3.3.24.1.2
<i>PE_UFP_VDM_Mode_Entry_NAK</i>	Section 8.3.3.24.1.3
UFP Structured VDM Exit Mode	
<i>PE_UFP_VDM_Mode_Exit</i>	Section 8.3.3.24.2.1
<i>PE_UFP_VDM_Mode_Exit_ACK</i>	Section 8.3.3.24.2.2
<i>PE_UFP_VDM_Mode_Exit_NAK</i>	Section 8.3.3.24.2.3

Table 8.154 Policy Engine States

State name	Reference
Cable Plug Specific	
Cable Ready	
<i>PE_CBL_Ready</i>	Section 8.3.3.25.1.1
Mode Entry	
<i>PE_CBL_Evaluate_Mode_Entry</i>	Section 8.3.3.25.4.1.1
<i>PE_CBL_Mode_Entry_ACK</i>	Section 8.3.3.25.4.1.2
<i>PE_CBL_Mode_Entry_NAK</i>	Section 8.3.3.25.4.1.3
Mode Exit	
<i>PE_CBL_Mode_Exit</i>	Section 8.3.3.25.4.2.1
<i>PE_CBL_Mode_Exit_ACK</i>	Section 8.3.3.25.4.2.2
<i>PE_CBL_Mode_Exit_NAK</i>	Section 8.3.3.25.4.1.3
Cable Soft Reset	
<i>PE_CBL_Soft_Reset</i>	Section 8.3.3.25.2.1.1
Cable Hard Reset	
<i>PE_CBL_Hard_Reset</i>	Section 8.3.3.25.2.2.1
DFP/VCONN Source Soft Reset or Cable Reset	
<i>PE_DFP_VCS_CBL_Send_Soft_Reset</i>	Section 8.3.3.25.2.3.1
<i>PE_DFP_VCS_CBL_Send_Cable_Reset</i>	Section 8.3.3.25.2.3.2
UFP/VCONN Source Soft Reset or Cable Reset	
<i>PE_UFP_VCS_CBL_Send_Soft_Reset</i>	Section 8.3.3.25.2.4.1
Source Startup Structured VDM Discover Identity	
<i>PE_SRC_VDM_Identity_Request</i>	Section 8.3.3.25.3.1
<i>PE_SRC_VDM_Identity_ACKed</i>	Section 8.3.3.25.3.2
<i>PE_SRC_VDM_Identity_NAKed</i>	Section 8.3.3.25.3.3
EPR Mode	
Source EPR Mode Entry	
<i>PE_SRC_Evaluate_EPR_Mode_Entry</i>	Section 8.3.3.26.1.1
<i>PE_SRC_EPR_Mode_Entry_Ack</i>	Section 8.3.3.26.1.2
<i>PE_SRC_EPR_Mode_Discover_Cable</i>	Section 8.3.3.26.1.3
<i>PE_SRC_EPR_Mode_Evaluate_Cable_EPR</i>	Section 8.3.3.26.1.4
<i>PE_SRC_EPR_Mode_Entry_Succeeded</i>	Section 8.3.3.26.1.5
<i>PE_SRC_EPR_Mode_Entry_Failed</i>	Section 8.3.3.26.1.6
Sink EPR Mode Entry	
<i>PE_SNK_Send_EPR_Mode_Entry</i>	Section 8.3.3.26.2.1
<i>PE_SNK_EPR_Mode_Wait_For_Response</i>	Section 8.3.3.26.2.2
Source EPR Mode Exit	
<i>PE_SRC_Send_EPR_Mode_Exit</i>	Section 8.3.3.26.3.1
<i>PE_SRC_EPR_Mode_Exit_Received</i>	Section 8.3.3.26.3.2

Table 8.154 Policy Engine States

State name	Reference
Sink EPR Mode Exit	
<i>PE_SNK_Send_EPR_Mode_Exit</i>	Section 8.3.3.26.4.1
<i>PE_SNK_EPR_Mode_Exit_Received</i>	Section 8.3.3.26.4.2
BIST	
BIST Carrier Mode	
<i>PE_BIST_Carrier_Mode</i>	Section 8.3.3.27.1.1
BIST Carrier Mode	
<i>PE_BIST_Test_Mode</i>	Section 8.3.3.27.2.1
BIST Shared Capacity Test Mode	
<i>PE_BIST_Shared_Capacity_Test_Mode</i>	Section 8.3.3.27.3.1
USB Type-C referenced states	
<i>ErrorRecovery</i>	Section 8.3.3.28.1

9 States and Status Reporting

9.1 Overview

This chapter describes the Status reporting mechanisms for devices with data connections (e.g., D+/D- and or SSTx+/- and SSRx+/-). It also describes the corresponding USB state a device that supports USB PD **Shall** transition to as a result of changes to the USB PD state that the device is in.

This chapter does not define the *System Policy* or the *System Policy Manager*. That is defined in [\[UCSI\]](#). In addition, the Policies themselves are not described here; these are left to the implementers of the relevant products and systems to define.

All PD Capable USB (PDUSB) *Devices* **Shall** report themselves as self-powered devices (over USB) when plugged into a PD capable *Port* even if they are entirely powered from *VBUS*. However, there are some differences between PD and [\[USB 2.0\]](#) / [\[USB 3.2\]](#); for example, the presence of *VBUS* alone does not mean that the device (*Consumer*) moves from the *USB Attached State* to the *USB Powered State*. Similarly, the removal of *VBUS* alone does not move the device (*Consumer*) from any of the USB states to the *USB Attached State*. See [Section 9.1.2, "Mapping to USB Device States"](#) for details.

PDUSB Devices **Shall** follow the PD requirements when it comes to suspend (see [Section 6.4.1.2.1.2, "USB Suspend Supported"](#)), configured, and operational power. The PD requirements when the device is configured or operational are defined in this section (see [Table 9.4, "PD Consumer Port Descriptor"](#)).

Note: The power requirements reported in the PD *Consumer Port* descriptor of the device **Shall** override the power draw reported in the *bMaxPower* field in the configuration descriptor. A *PDUSB Device* **Shall** report zero in the *bMaxPower* field after successfully negotiating a mutually agreeable *Explicit Contract* and **Shall** disconnect and re-enumerate when it switches operation back to operating in standard [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB4\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#). When operating in [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#) mode it **Shall** report its power draw via the *bMaxPower* field.

Each *Provider* and *Consumer* will have their own Local Policies which operate between *Port Partners*. An example of a typical PD system is shown in [Figure 9.1, "Example PD Topology"](#). This example consists of a *Provider*, *Consumer/Providers* and *Consumers* connected together in a tree topology. Between directly connected devices there is both a flow of Power and also Communication consisting of both Status and Control information.

Figure 9.1 Example PD Topology

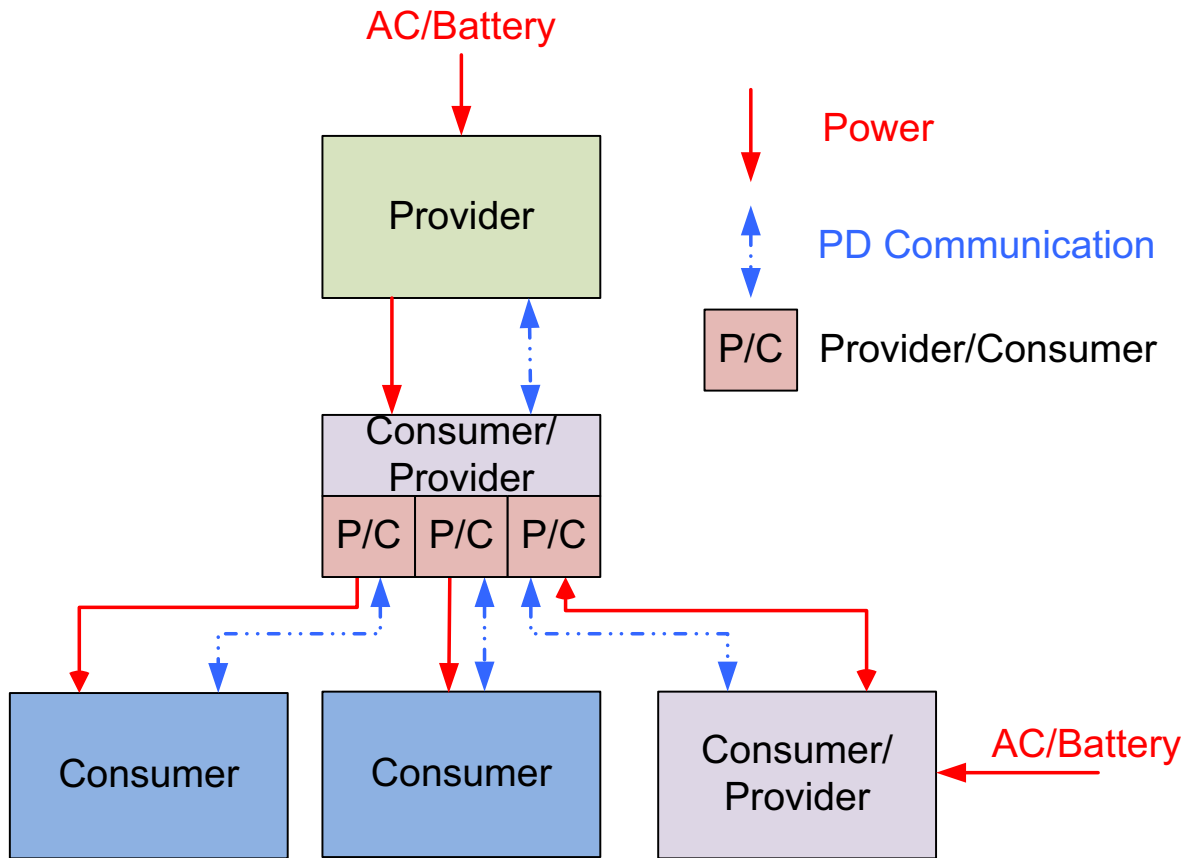
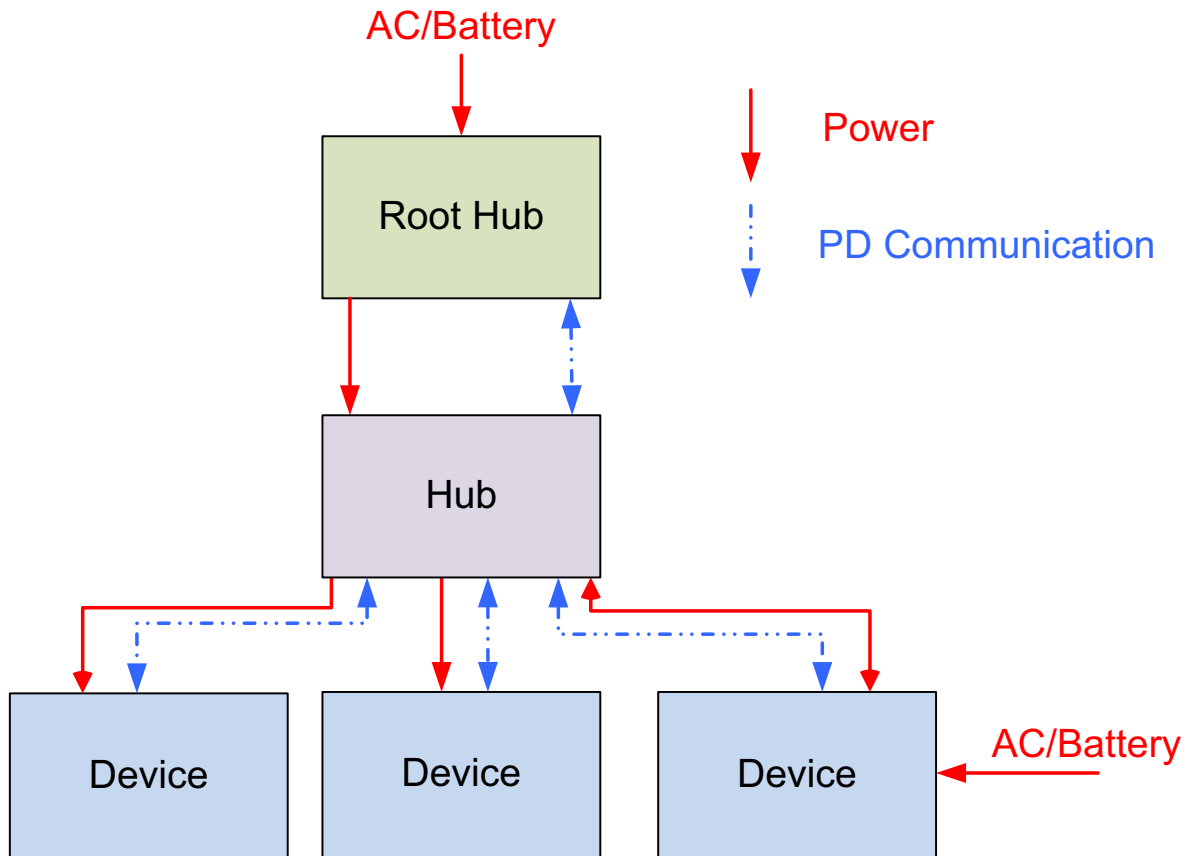


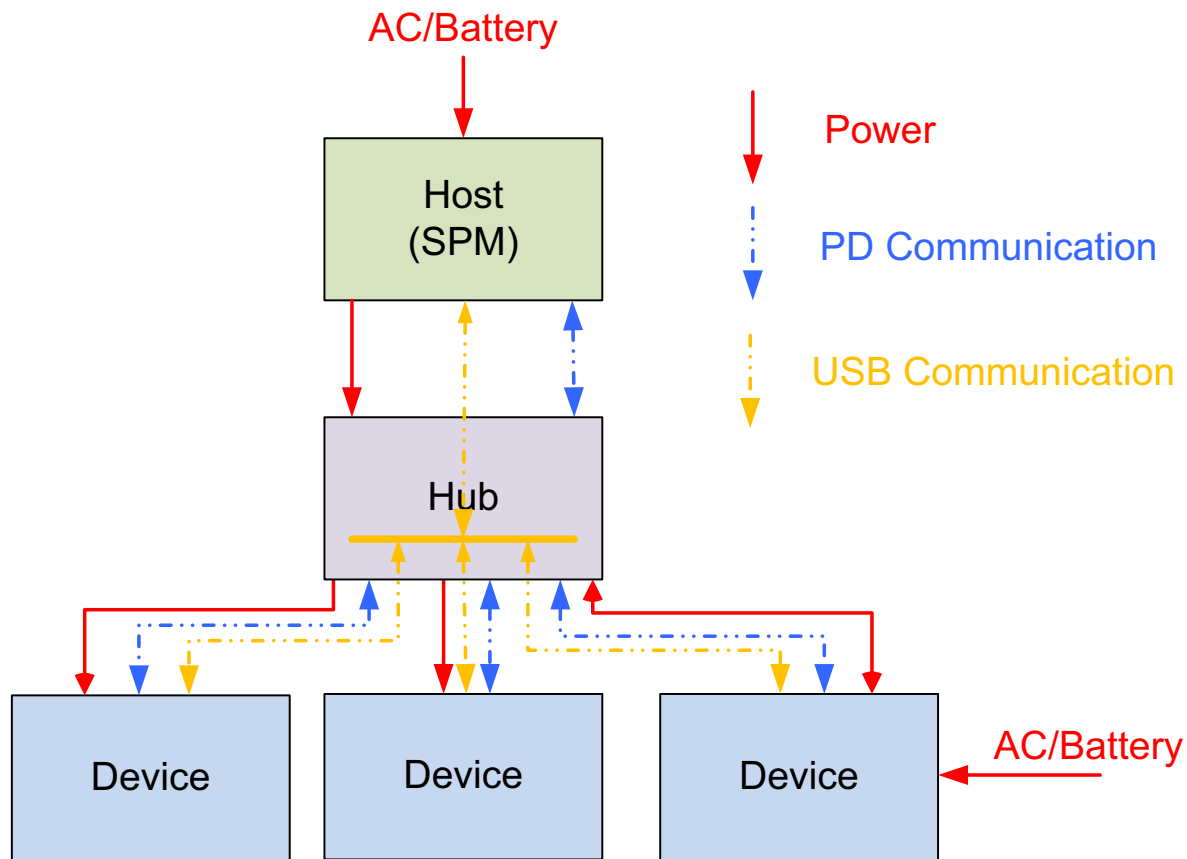
Figure 9.2, "Mapping of PD Topology to USB" shows how this same topology can be mapped to USB.

Figure 9.2 Mapping of PD Topology to USB



In a USB based system, policy is managed by the host and communication of system level policy information is via standard USB data line communication. This is a separate mechanism to the USB Power Delivery *VBUS* protocol which is used to manage *Local Policy*. When *USB Communication* is used, status information and control requests are passed directly between the *System Policy Manager (SPM)* on the host and the *Provider* or *Consumer*. See [Figure 9.3, "Use of SPM in the PD System"](#).

Figure 9.3 Use of SPM in the PD System



Status information comes from a *Provider* or *Consumer* to the *SPM* so it can better manage the resources on the host and provide feedback to the end user.

Real systems will be a mixture of devices which in terms of power management support might have implemented PD, [\[USB 2.0\]](#), [\[USB 3.2\]](#), [\[USB4\]](#), [\[USB Type-C 2.4\]](#) or [\[USBBC 1.2\]](#) or they might even just be non-compliant “power sucking devices”. The level of communication of system status to the *SPM* will therefore not necessarily be comprehensive. The aim of the status mechanisms described here is to provide a mechanism whereby each connected entity in the system provides as much information as possible on the status of itself.

Information described in this section that is communicated to the *SPM* is as follows:

- Versions of *USB Type-C*[®], PD and BC supported.
- Capabilities as a *Provider/Consumer*.
- Current operational state of each *Port* e.g. Standard, *USB Type-C* Current, BC, PD and *Negotiated* power level.
- Status of AC or *Battery* Power for each *PDUSB Device* in the system.

The *SPM* can *Negotiate* with *Providers* or *Consumers* in the system in order to request a different *Local Policy*, or to request the amount of power to be delivered by the *Provider* to the *Consumer*. Any change in *Local Policy* could

trigger a *Re-negotiation* of the *Explicit Contract*, using USB Power Delivery protocols, between a directly connected *Provider* and *Consumer*. A change in how much power is available can, for example, cause a *Re-negotiation*.

9.1.1 PDUSB Device and Hub Requirements

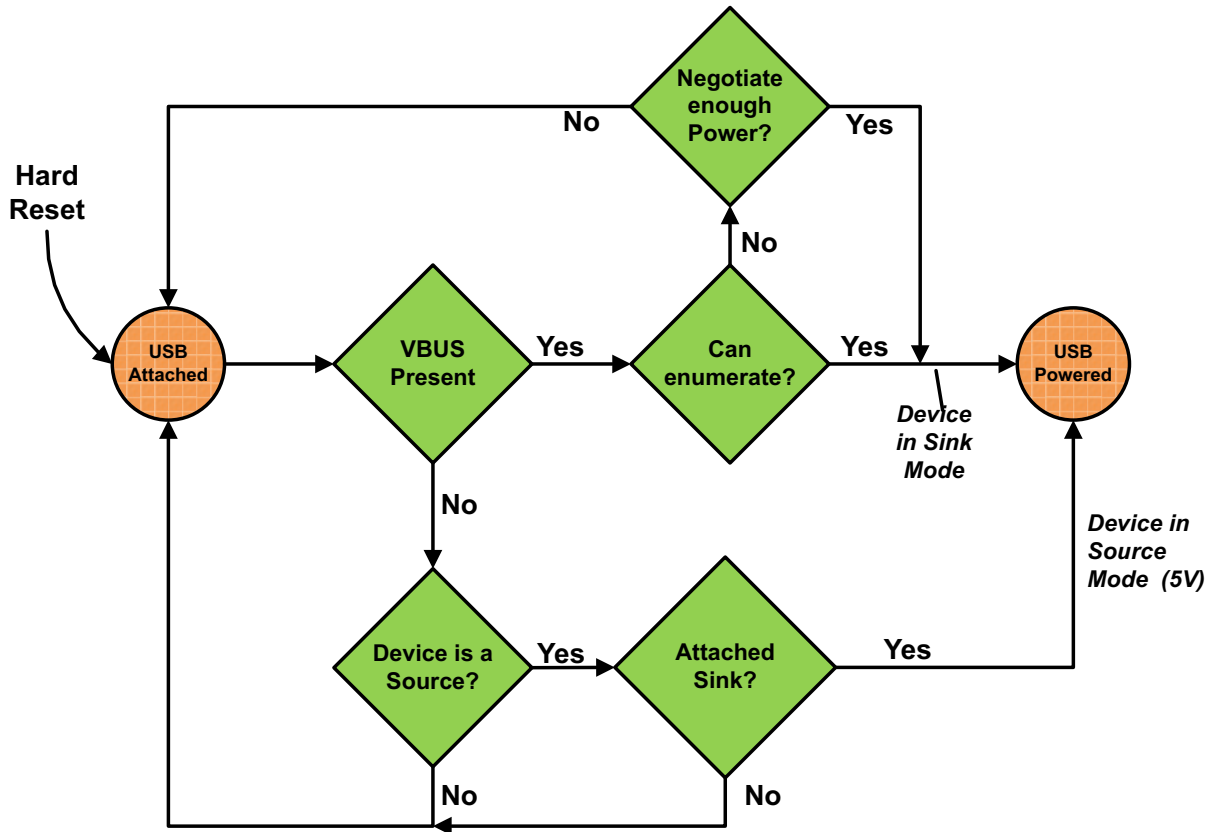
All *PDUSB Devices* **Shall** return all relevant descriptors mentioned in this chapter. *PDUSB Hubs* **Shall** also support a PD bridge as defined in [\[UCSI\]](#).

9.1.2 Mapping to USB Device States

As mentioned in [Section 9.1, "Overview"](#) a *PDUSB Device* reports itself as a self-powered device. However, the device **Shall** determine whether or not it is in the *USB Attached State* or *USB Powered State* as described in [Figure 9.4, "USB Attached to USB Powered State Transition"](#), [Figure 9.5, "Any USB State to USB Attached State Transition \(When operating as a Consumer\)"](#) and [Figure 9.6, "Any USB State to USB Attached State Transition \(When operating as a Provider\)"](#). All other USB states of the *PDUSB Device* **Shall** be as described in Chapter 9 of [\[USB 2.0\]](#) and [\[USB 3.2\]](#).

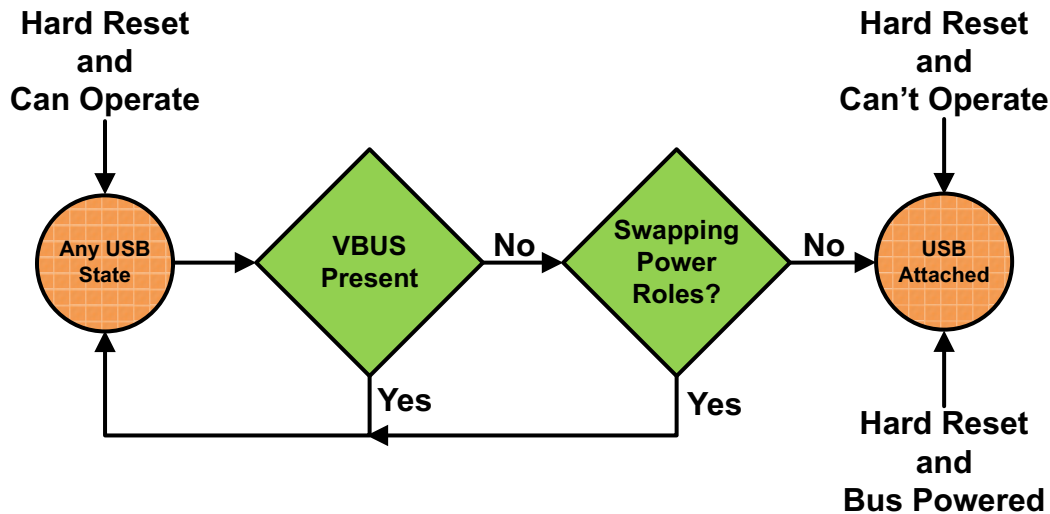
[Figure 9.4, "USB Attached to USB Powered State Transition"](#) shows how a *PDUSB Device* determines when to transition from the *USB Attached State* to the *USB Powered State*. *USB Type-C Dead Battery* operation does not require special handling since the default state at *Attach* or after a *Hard Reset* is that the *USB Device* is a *Sink*.

Figure 9.4 USB Attached to USB Powered State Transition



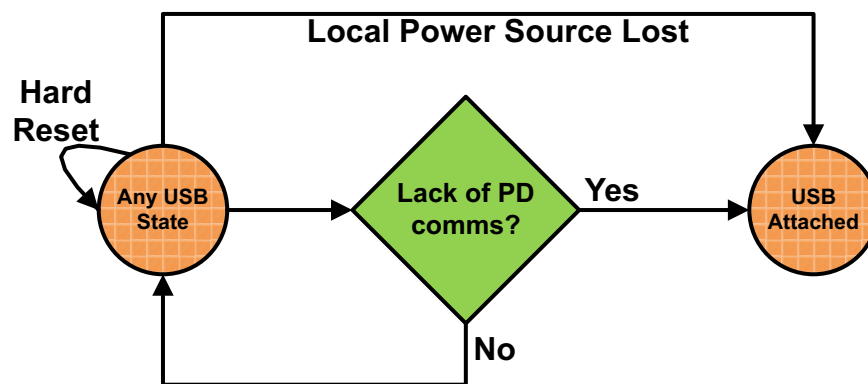
[Figure 9.5, "Any USB State to USB Attached State Transition \(When operating as a Consumer\)"](#) shows how a *PDUSB Device* determines when to transition from the *USB Powered State* to the *USB Attached State* when the device is a *Consumer*. A *PDUSB Device* determines that it is performing a *Power Role Swap* as described in [Section 8.3.3.19.3, "Policy Engine in Source to Sink Power Role Swap State Diagram"](#) and [Section 8.3.3.19.4, "Policy Engine in Sink to Source Power Role Swap State Diagram"](#). See [Section 7.1.5, "Response to Hard Resets"](#) for additional information on device behavior during *Hard Resets*.

Figure 9.5 Any USB State to USB Attached State Transition (When operating as a Consumer)



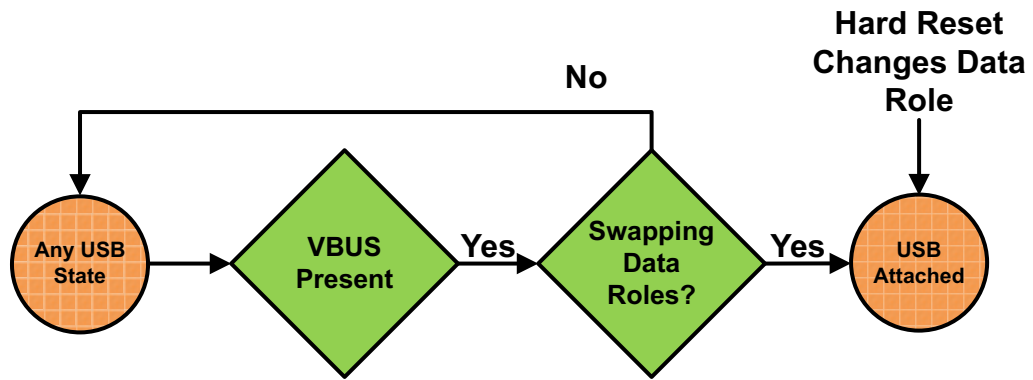
[Figure 9.6, "Any USB State to USB Attached State Transition \(When operating as a Provider\)"](#) shows how a PDUSB Device determines when to transition from the USB Powered State to the USB Attached State when the device is a Provider.

Figure 9.6 Any USB State to USB Attached State Transition (When operating as a Provider)



[Figure 9.7, "Any USB State to USB Attached State Transition \(After a USB Type-C Data Role Swap\)"](#) shows how a PDUSB Device using the USB Type-C connector determines when to transition from the USB Powered State to the USB Attached State after a Data Role Swap has been performed i.e., it has just changed from operation as a PDUSB Host to operation as a PDUSB Device. The Data Role Swap is described in [Section 6.3.9, "DR Swap Message"](#). A Hard Reset will also return a Sink acting as a PDUSB Host to PDUSB Device operation as described in [Section 6.8.3, "Hard Reset"](#). See [Section 7.1.5, "Response to Hard Resets"](#) for additional information on device behavior during Hard Resets.

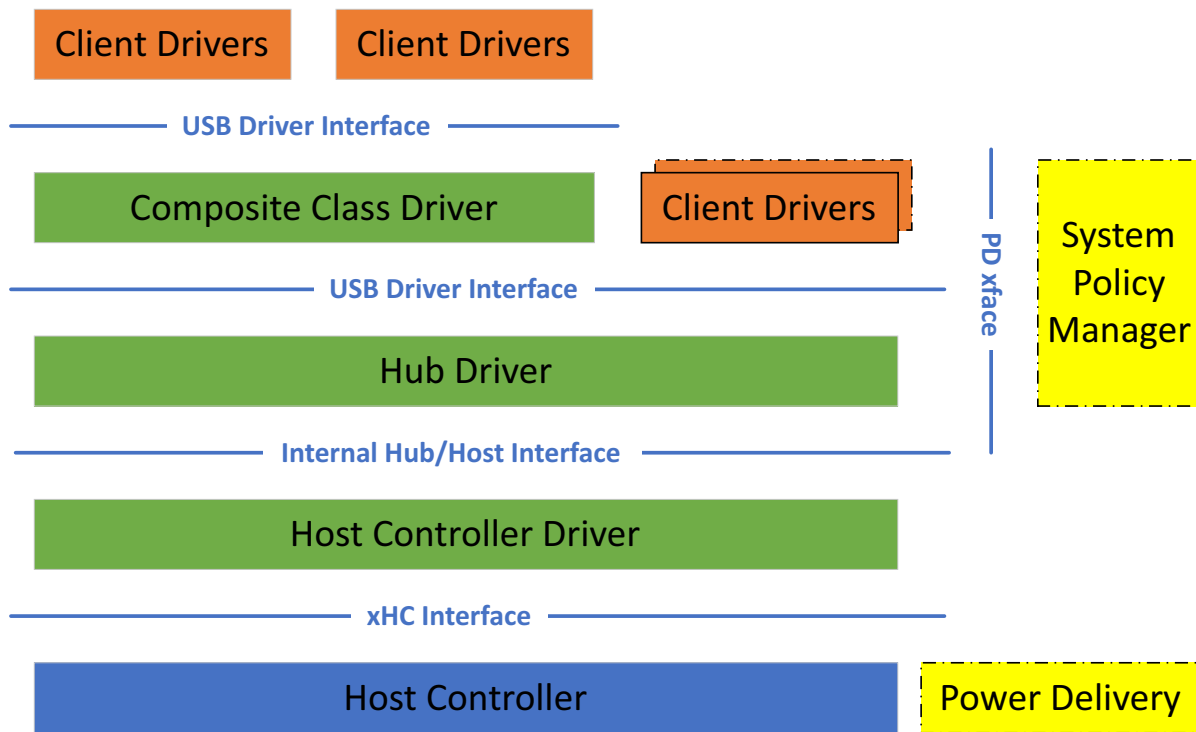
Figure 9.7 Any USB State to USB Attached State Transition (After a USB Type-C Data Role Swap)



9.1.3 PD Software Stack

[Figure 9.8, "Software stack on a PD aware OS"](#) gives an example of the software stack on a PD aware OS. In this stack we are using the example of a system with an xHCI based controller. The USB Power Delivery hardware **May** or **May Not** be a part of the xHC.

Figure 9.8 Software stack on a PD aware OS

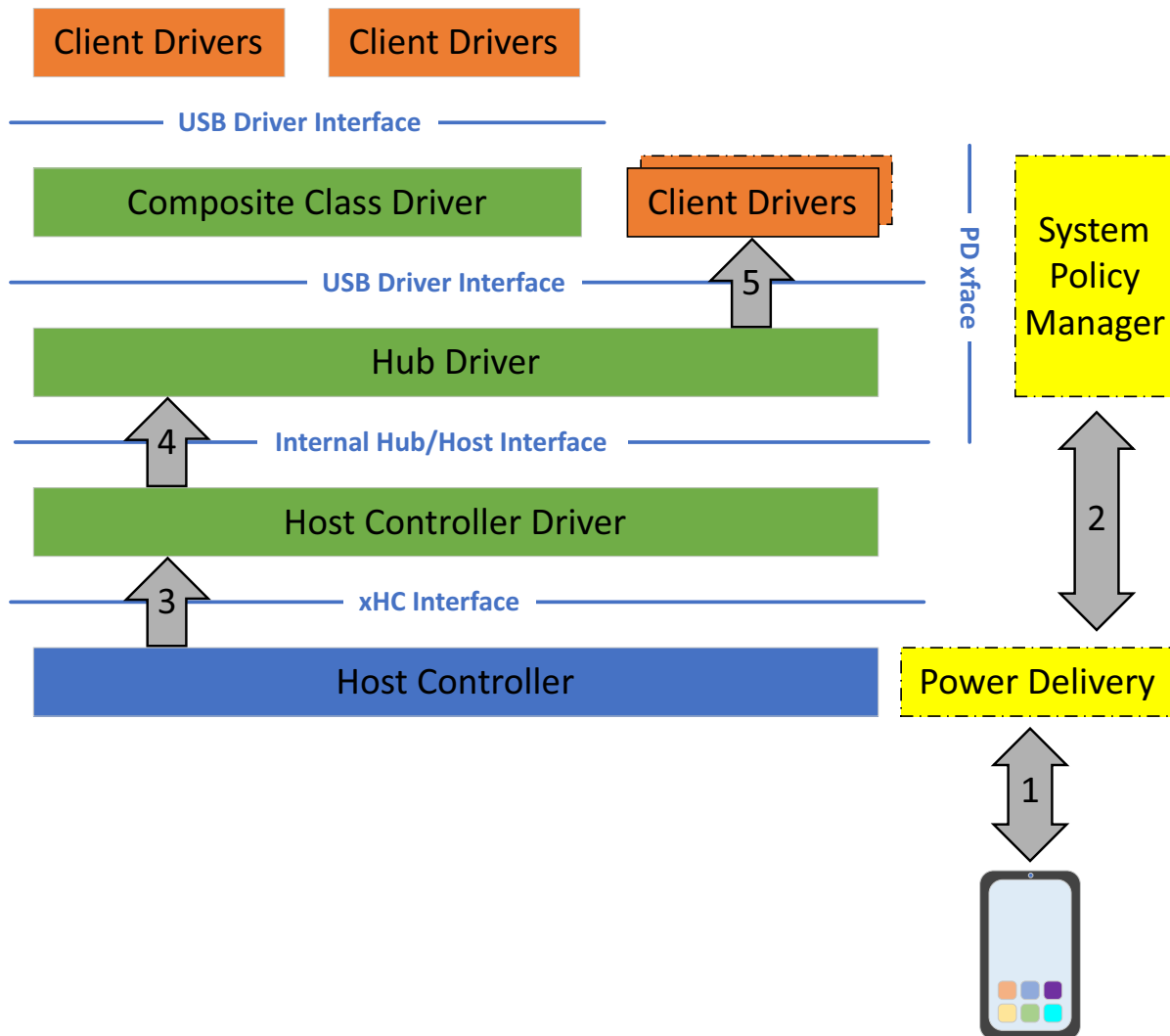


9.1.4 PDUSB Device Enumeration

As described earlier, a *PDUSB Device* acts as a self-powered device with some caveats with respect to how it transitions from the *USB Attached State* to *USB Powered State*. [Figure 9.9, "Enumeration of a PDUSB Device"](#) gives a high-level overview of the enumeration steps involved due to this change. A *PDUSB Device* will first (Step1) interact with the Power Delivery hardware and the *Local Policy* manager to determine whether or not it can get sufficient power to enumerate/operate.

PD is likely to have established a *Explicit Contract* prior to enumeration. The *SPM* will be notified (Step 2) of the result of this *Negotiation* between the Power Delivery hardware and the *PDUSB Device*. After successfully negotiating a mutually agreeable *Explicit Contract* the device will signal a connect to the xHC. The standard USB enumeration process (Steps 3, 4 and 5) is then followed to load the appropriate driver for the function(s) that the *PDUSB Device* exposes.

Figure 9.9 Enumeration of a PDUSB Device



If a *PDUSB Device* cannot perform its intended function with the amount of power that it can get from the *Port* it is connected to, then the host system **Should** display a notification (on a PD aware OS) about the failure to provide sufficient power to the device. In addition, the device **Shall** follow the requirements listed in [Section 8.2.5.2.1, "Local device handling of mismatch"](#).

9.2 PD Specific Descriptors

A *PDUSB Device* **Shall** return all relevant descriptors mentioned in this section.

The device **Shall** return its capability descriptors as part of the device's Binary Object Store (BOS) descriptor set. [Table 9.1, "USB Power Delivery Type Codes"](#) lists the type of PD device capabilities.

Table 9.1 USB Power Delivery Type Codes

Capability Code	Value	Description
<i>POWER_DELIVERY_CAPABILITY</i>	06H	Defines the various PD Capabilities of this device
<i>BATTERY_INFO_CAPABILITY</i>	07H	Provides information on each <i>Battery</i> supported by the device
<i>PD_CONSUMER_PORT_CAPABILITY</i>	08H	The <i>Consumer</i> characteristics of a <i>Port</i> on the device
<i>PD_PROVIDER_PORT_CAPABILITY</i>	09H	The <i>Provider</i> characteristics of a <i>Port</i> on the device

9.2.1 USB Power Delivery Capability Descriptor

Table 9.2, "USB Power Delivery Capability Descriptor" details the fields in the USB **POWER_DELIVERY_CAPABILITY** Descriptor.

Table 9.2 USB Power Delivery Capability Descriptor

Offset	Field	Size	Value	Description																																						
0	bLength	1	Number	Size of descriptor																																						
1	bDescriptorType	1	Constant	DEVICE CAPABILITY Descriptor type																																						
2	bDevCapabilityType	1	Constant	Capability type: <i>POWER_DELIVERY_CAPABILITY</i>																																						
3	bReserved	1	<i>Reserved</i>	<i>Shall</i> be set to zero.																																						
4	bmAttributes	4	Bitmap	Bitmap encoding of supported device level features. A value of one in a bit location indicates a feature is supported; a value of zero indicates it is not supported. Encodings are:																																						
				<table><tr><th>Bit</th><th>Description</th></tr><tr><td>0</td><td><i>Reserved. Shall</i> be set to zero.</td></tr><tr><td>1</td><td>Battery Charging. This bit <i>Shall</i> be set to one to indicate this device supports <i>[USBBC 1.2]</i> as per the value reported in the bcdBCVersion field.</td></tr><tr><td>2</td><td>USB Power Delivery. This bit <i>Shall</i> be set to one to indicate this device supports the USB Power Delivery Specification as per the value reported in the bcdPDVersion field.</td></tr><tr><td>3</td><td><i>Provider</i>. This bit <i>Shall</i> be set to one to indicate this device is capable of providing power. This field is only <i>Valid</i> if Bit 2 is set to one.</td></tr><tr><td>4</td><td><i>Consumer</i>. This bit <i>Shall</i> be set to one to indicate that this device is a consumer of power. This field is only <i>Valid</i> if Bit 2 is set to one.</td></tr><tr><td>5</td><td>This bit <i>Shall</i> be set to 1 to indicate that this device supports the feature CHARGING_POLICY. Note: Supporting the CHARGING_POLICY feature does not require a BC or PD mechanism to be implemented.</td></tr><tr><td>6</td><td><i>USB Type-C Current</i>. This bit <i>Shall</i> be set to one to indicate this device supports power capabilities defined in<i>[USB Type-C 2.4]</i> as per the value reported in the bcdUSBTypeCVersion field</td></tr><tr><td>7</td><td><i>Reserved. Shall</i> be set to zero.</td></tr><tr><td>15:8</td><td>bmPowerSource. At least one of the following bits 8, 9 and 14 <i>Shall</i> be set to indicate which power sources are supported.</td></tr><tr><td></td><td><table><tr><th>Bit</th><th>Description</th></tr><tr><td>8</td><td><i>AC Supply</i></td></tr><tr><td>9</td><td><i>Battery</i></td></tr><tr><td>10</td><td>Other</td></tr><tr><td>13:11</td><td>NumBatteries. This field <i>Shall</i> only be <i>Valid</i> when the Battery field is set to one and <i>Shall</i> be used to report the number of batteries in the device.</td></tr><tr><td>14</td><td>Uses <i>VBUS</i></td></tr><tr><td>15</td><td><i>Reserved</i> and <i>Shall</i> be set to zero.</td></tr></table></td></tr><tr><td>13:16</td><td><i>Reserved. Shall</i> be set to zero.</td></tr></table>	Bit	Description	0	<i>Reserved. Shall</i> be set to zero.	1	Battery Charging. This bit <i>Shall</i> be set to one to indicate this device supports <i>[USBBC 1.2]</i> as per the value reported in the bcdBCVersion field.	2	USB Power Delivery. This bit <i>Shall</i> be set to one to indicate this device supports the USB Power Delivery Specification as per the value reported in the bcdPDVersion field.	3	<i>Provider</i> . This bit <i>Shall</i> be set to one to indicate this device is capable of providing power. This field is only <i>Valid</i> if Bit 2 is set to one.	4	<i>Consumer</i> . This bit <i>Shall</i> be set to one to indicate that this device is a consumer of power. This field is only <i>Valid</i> if Bit 2 is set to one.	5	This bit <i>Shall</i> be set to 1 to indicate that this device supports the feature CHARGING_POLICY. Note: Supporting the CHARGING_POLICY feature does not require a BC or PD mechanism to be implemented.	6	<i>USB Type-C Current</i> . This bit <i>Shall</i> be set to one to indicate this device supports power capabilities defined in <i>[USB Type-C 2.4]</i> as per the value reported in the bcdUSBTypeCVersion field	7	<i>Reserved. Shall</i> be set to zero.	15:8	bmPowerSource. At least one of the following bits 8, 9 and 14 <i>Shall</i> be set to indicate which power sources are supported.		<table><tr><th>Bit</th><th>Description</th></tr><tr><td>8</td><td><i>AC Supply</i></td></tr><tr><td>9</td><td><i>Battery</i></td></tr><tr><td>10</td><td>Other</td></tr><tr><td>13:11</td><td>NumBatteries. This field <i>Shall</i> only be <i>Valid</i> when the Battery field is set to one and <i>Shall</i> be used to report the number of batteries in the device.</td></tr><tr><td>14</td><td>Uses <i>VBUS</i></td></tr><tr><td>15</td><td><i>Reserved</i> and <i>Shall</i> be set to zero.</td></tr></table>	Bit	Description	8	<i>AC Supply</i>	9	<i>Battery</i>	10	Other	13:11	NumBatteries. This field <i>Shall</i> only be <i>Valid</i> when the Battery field is set to one and <i>Shall</i> be used to report the number of batteries in the device.	14	Uses <i>VBUS</i>	15	<i>Reserved</i> and <i>Shall</i> be set to zero.	13:16	<i>Reserved. Shall</i> be set to zero.
Bit				Description																																						
0				<i>Reserved. Shall</i> be set to zero.																																						
1				Battery Charging. This bit <i>Shall</i> be set to one to indicate this device supports <i>[USBBC 1.2]</i> as per the value reported in the bcdBCVersion field.																																						
2				USB Power Delivery. This bit <i>Shall</i> be set to one to indicate this device supports the USB Power Delivery Specification as per the value reported in the bcdPDVersion field.																																						
3				<i>Provider</i> . This bit <i>Shall</i> be set to one to indicate this device is capable of providing power. This field is only <i>Valid</i> if Bit 2 is set to one.																																						
4				<i>Consumer</i> . This bit <i>Shall</i> be set to one to indicate that this device is a consumer of power. This field is only <i>Valid</i> if Bit 2 is set to one.																																						
5				This bit <i>Shall</i> be set to 1 to indicate that this device supports the feature CHARGING_POLICY. Note: Supporting the CHARGING_POLICY feature does not require a BC or PD mechanism to be implemented.																																						
6				<i>USB Type-C Current</i> . This bit <i>Shall</i> be set to one to indicate this device supports power capabilities defined in <i>[USB Type-C 2.4]</i> as per the value reported in the bcdUSBTypeCVersion field																																						
7				<i>Reserved. Shall</i> be set to zero.																																						
15:8				bmPowerSource. At least one of the following bits 8, 9 and 14 <i>Shall</i> be set to indicate which power sources are supported.																																						
				<table><tr><th>Bit</th><th>Description</th></tr><tr><td>8</td><td><i>AC Supply</i></td></tr><tr><td>9</td><td><i>Battery</i></td></tr><tr><td>10</td><td>Other</td></tr><tr><td>13:11</td><td>NumBatteries. This field <i>Shall</i> only be <i>Valid</i> when the Battery field is set to one and <i>Shall</i> be used to report the number of batteries in the device.</td></tr><tr><td>14</td><td>Uses <i>VBUS</i></td></tr><tr><td>15</td><td><i>Reserved</i> and <i>Shall</i> be set to zero.</td></tr></table>	Bit	Description	8	<i>AC Supply</i>	9	<i>Battery</i>	10	Other	13:11	NumBatteries. This field <i>Shall</i> only be <i>Valid</i> when the Battery field is set to one and <i>Shall</i> be used to report the number of batteries in the device.	14	Uses <i>VBUS</i>	15	<i>Reserved</i> and <i>Shall</i> be set to zero.																								
Bit				Description																																						
8				<i>AC Supply</i>																																						
9				<i>Battery</i>																																						
10				Other																																						
13:11	NumBatteries. This field <i>Shall</i> only be <i>Valid</i> when the Battery field is set to one and <i>Shall</i> be used to report the number of batteries in the device.																																									
14	Uses <i>VBUS</i>																																									
15	<i>Reserved</i> and <i>Shall</i> be set to zero.																																									
13:16	<i>Reserved. Shall</i> be set to zero.																																									

Table 9.2 USB Power Delivery Capability Descriptor

Offset	Field	Size	Value	Description
8	bcdBCVersion	2	BCD	Battery Charging Specification Release Number in Binary-Coded Decimal (e.g., V1.20 is 120H). This field shall only be Valid if the device indicates that it supports [USBBC 1.2] in the bmAttributes field.
10	bcdPDVersion	2	BCD	USB Power Delivery Specification Release Number in Binary-Coded Decimal. This field shall only be Valid if the device indicates that it supports PD in the bmAttributes field.
12	bcdUSBTypeCVersion	2	BCD	USB Type-C Specification Release Number in Binary-Coded Decimal. This field shall only be Valid if the device indicates that it supports USB Type-C in the bmAttributes field.

9.2.2 Battery Info Capability Descriptor

A PDUSB Device **shall** support the capability descriptor shown in [Table 9.3, "Battery Info Capability Descriptor"](#) if it reported that one of its power sources was a *Battery* in the bmPowerSource field in its Power Deliver Capability Descriptor. It **shall** return one **BATTERY_INFO_CAPABILITY** Descriptor per *Battery* it supports.

Table 9.3 Battery Info Capability Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of descriptor
1	bDescriptorType	1	Constant	DEVICE CAPABILITY Descriptor type
2	bDevCapabilityType	1	Constant	Capability type: BATTERY_INFO_CAPABILITY
3	iBattery	1	Index	Index of string descriptor shall contain the user-friendly name for this <i>Battery</i> .
4	iSerial	1	Index	Index of string descriptor shall contain the Serial Number String for this <i>Battery</i> .
5	iManufacturer	1	Index	Index of string descriptor shall contain the name of the Manufacturer for this <i>Battery</i> .
6	bBatteryId	1	Number	Value shall be used to uniquely identify this <i>Battery</i> in status Messages.
7	bReserved	1	Number	Reserved and shall be set to zero.
8	dwChargedThreshold	4	mWh	shall contain the <i>Battery</i> charge value above which this <i>Battery</i> is considered to be fully charged but not necessarily "topped off."
12	dwWeakThreshold	4	mWh	shall contain the minimum charge level of this <i>Battery</i> such that above this threshold, a device can be assured of being able to power up successfully (see [USBBC 1.2]).
16	dwBatteryDesignCapacity	4	mWh	shall contain the design capacity of the <i>Battery</i> .
20	dwBatteryLastFullchargeCapacity	4	mWh	shall contain the maximum capacity of the <i>Battery</i> when fully charged.

9.2.3 PD Consumer Port Capability Descriptor

A PDUSB Device **Shall** support the **PD_CONSUMER_PORT_CAPABILITY** descriptor shown in [Table 9.4, "PD Consumer Port Descriptor"](#) if it is a *Consumer*.

Table 9.4 PD Consumer Port Descriptor

Offset	Field	Size	Value	Description	
0	bLength	1	Number	Size of descriptor	
1	bDescriptorType	1	Constant	DEVICE CAPABILITY Descriptor type	
2	bDevCapabilityType	1	Constant	Capability type: <i>PD_CONSUMER_PORT_CAPABILITY</i>	
3	bReserved	1	Number	<i>Reserved</i> and <i>Shall</i> be set to zero.	
4	bmCapabilities	2	Bitmap	Capability: This field <i>Shall</i> indicate the specification the <i>Consumer Port</i> will operate under.	
				Bit	Description
				0	Battery Charging (BC)
				1	USB Power Delivery (PD)
				2	<i>USB Type-C</i> Current
				15:3	<i>Reserved</i> and <i>Shall</i> be set to zero.
6	wMinVoltage	2	Number	<i>Shall</i> contain the minimum voltage in 50mV units that this <i>Consumer</i> is capable of operating at.	
8	wMaxVoltage	2	Number	<i>Shall</i> contain the maximum voltage in 50mV units that this <i>Consumer</i> is capable of operating at.	
10	wReserved	2	Number	<i>Reserved</i> and <i>Shall</i> be set to zero.	
12	dwMaxOperatingPower	4	Number	<i>Shall</i> contain the maximum power in 10mW units this <i>Consumer</i> can draw when it is in a steady state operating mode.	
16	dwMaxPeakPower	4	Number	<i>Shall</i> contain the maximum power in 10mW units this <i>Consumer</i> can draw for a short duration of time (<i>dwMaxPeakPowerTime</i>) before it falls back into a steady state.	
20	dwMaxPeakPowerTime	4	Number	<i>Shall</i> contain the time in 100ms units that this <i>Consumer</i> can draw peak current. A device <i>Shall</i> set this field to 0xFFFF if this value is unknown.	

9.2.4 PD Provider Port Capability Descriptor

A PDUSB Device **shall** support the **PD_PROVIDER_PORT_CAPABILITY** descriptor shown in [Table 9.5, "PD Provider Port Descriptor"](#) if it is a *Provider*.

Table 9.5 PD Provider Port Descriptor

Offset	Field	Size	Value	Description	
0	bLength	1	Number	Size of descriptor	
1	bDescriptorType	1	Constant	DEVICE CAPABILITY Descriptor type	
2	bDevCapabilityType	1	Constant	Capability type: <i>PD_PROVIDER_PORT_CAPABILITY</i>	
3	bReserved	1	Number	Reserved and Shall be set to zero.	
4	bmCapabilities	2	Bitmap	Capability: This field Shall indicate the specification the <i>Provider Port</i> will operate under.	
				Bit	Description
				0	Battery Charging (BC)
				1	USB Power Delivery (PD)
				2	<i>USB Type-C</i> Current
				15:3	Reserved and Shall be set to zero.
6	bNumOfPDObjects	1	Number	Shall indicate the number of <i>Power Data Objects</i> .	
7	bReserved	1	Number	Reserved and Shall be set to zero.	
8	wPowerDataObject1	4	Bitmap	Shall contain the first <i>Power Data Object</i> supported by this <i>Provider Port</i> . See Section 6.4.1, "Capabilities Message" for details of the <i>Power Data Objects</i> .	
...	
4*(N+1)	wPowerDataObjectN	4	Bitmap	Shall contain the 2nd and subsequent <i>Power Data Objects</i> supported by this <i>Provider Port</i> . See Section 6.4.1, "Capabilities Message" for details of the <i>Power Data Objects</i> .	

9.3 PD Specific Requests and Events

A *PDUSB Device* that is compliant to this specification **Shall** support the *Battery* related requests if it has a *Battery*.

A *PDUSB Hub* that is compliant to this specification **Shall** support a USB PD Bridge as described in [\[UCSI\]](#) irrespective of whether the *PDUSB Hub* is a *Provider*, a *Consumer*, or both.

9.3.1 PD Specific Requests

PD defines requests to which *PDUSB Devices* **Shall** respond as outlined in [Table 9.6, "PD Requests"](#). All **Valid** requests in [Table 9.6, "PD Requests"](#) **Shall** be implemented by *PDUSB Devices*.

Table 9.6 PD Requests

Request	bmRequestType	bRequest	wValue	wIndex	wLength	Data
GetBatteryStatus	10000000B	<i>GET_BATTERY_STATUS</i>	Zero	Battery ID	Eight	Battery Status
SetPDFeature	00000000B	set_feature	Feature Selector	Feature Specific	Zero	None

[Table 9.7, "PD Request Codes"](#) gives the bRequest values for *Commands* that are not listed in the hub/device framework chapters of [\[USB 2.0\]](#), [\[USB 3.2\]](#).

Table 9.7 PD Request Codes

bRequest	Value
<i>GET_BATTERY_STATUS</i>	21

[Table 9.8, "PD Feature Selectors"](#) gives the **Valid** feature selectors for the PD class. Refer to [Section 9.4.2.1, "BATTERY_WAKE_MASK Feature Selector"](#), and [Section 9.4.2.2, "CHARGING_POLICY Feature Selector"](#) for a description of the features.

Table 9.8 PD Feature Selectors

Feature Selector	Recipient	Value
<i>BATTERY_WAKE_MASK</i>	<i>Device</i>	40
<i>CHARGING_POLICY</i>	<i>Device</i>	54

9.4 PDUSB Hub and PDUSB Peripheral Device Requests

9.4.1 GetBatteryStatus

The request shown in [Table 9.9, "Get Battery Status Request"](#) returns the current status of the *Battery* in a *PDUSB Hub/Peripheral*, with *Battery* Status information as shown in [Table 9.10, "Battery Status Structure"](#).

Table 9.9 Get Battery Status Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	<i>GET_BATTERY_STATUS</i>	Zero	Battery ID	Eight	Battery Status

Table 9.10 Battery Status Structure

Offset	Field	Size	Value	Description	
0	bBatteryAttributes	1	Number	Shall indicate whether a <i>Battery</i> is installed and whether this is charging or discharging.	
				Value	Description
				0	There is no <i>Battery</i>
				1	The <i>Battery</i> is charging
				2	The <i>Battery</i> is discharging
				3	The <i>Battery</i> is neither discharging nor charging
			255...4	Reserved and Shall Not be used	
1	bBatterySOC	1	Number	Shall indicate the Battery State of Charge given as percentage value from Battery Remaining Capacity.	
2	bBatteryStatus	1	Number	If a <i>Battery</i> is present Shall indicate the present status of the <i>Battery</i> .	
				Value	Description
				0	No error
				1	<i>Battery</i> required and not present
				2	<i>Battery</i> non-chargeable/wrong chemistry
				3	Over-temp shutdown
				4	Over-voltage shutdown
				5	Over-current shutdown
				6	Fatigued <i>Battery</i>
				7	Unspecified error
	255...8	Reserved and Shall Not be used			

Table 9.10 Battery Status Structure

Offset	Field	Size	Value	Description	
3	bRemoteWakeCapStatus	1	Bitmap	If the device supports remote wake, then the device Shall support Battery Remote wake events. The default value for the Remote wake events Shall be turned off (set to zero) and can be enable/disabled by the host as required. If set to one the device Shall generate a wake event when a change of status occurs. See Section 9.4.2, "SetPDFeature" for more details.	
				Value	Description
				0	Battery present event
				1	Charging flow
				2	Battery error
7:3	Reserved and Shall be set to zero				
4	wRemainingOperatingTime	2	Number	<p>Shall contain the operating time (in minutes) until the Weak Battery threshold is reached, based on Present Battery Strength and the device's present operational power needs.</p> <p>Note: This value Shall exclude any additional power received from charging.</p> <p>A Battery that is not capable of returning this information Shall return a value of 0xFFFF.</p>	
6	wRemainingChargeTime	2	Number	<p>Shall contain the remaining time (in minutes) until the Charged Battery threshold is reached based on Present Battery Strength, charging power and the device's present operational power needs. Value Shall only be Valid if the Charging Flow is "Charging".</p> <p>A Battery that is not capable of returning this information Shall return a value of 0xFFFF.</p>	

If wValue or wLength are not as specified above, then the behavior of the *PDUSB Device* is not specified.

If wIndex refers to a Battery that does not exist, then the *PDUSB Device* **Shall** respond with a Request Error.

If the *PDUSB Device* is not configured, the *PDUSB Hub*'s response to this request is undefined.

If the *PDUSB Hub* is not configured, the *PDUSB Hub*'s response to this request is undefined.

9.4.2 SetPDFeature

The request shown in [Table 9.11, "Set PD Feature"](#) sets the value requested in the *PDUSB Hub/Peripheral*.

Table 9.11 Set PD Feature

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	set_feature	Feature Selector	Feature Specific	Zero	None

Setting a feature enables that feature or starts a process associated with that feature; see [Table 9.8, "PD Feature Selectors"](#) for the feature selector definitions. Features that **May** be set with this request are:

- BATTERY_WAKE_MASK.
- CHARGING_POLICY.

9.4.2.1 BATTERY_WAKE_MASK Feature Selector

When the feature selector is set to BATTERY_WAKE_MASK, then the wIndex field is structured as shown in [Table 9.12, "Battery Wake Mask"](#).

Table 9.12 Battery Wake Mask

Bit	Description
0	Battery Present: When this bit is set then the <i>PDUSB Device</i> Shall generate a wake event if it detects that a Battery has been inserted.
1	Charging Flow: When this bit is set then the <i>PDUSB Device</i> Shall generate a wake event if it detects that a Battery switched from charging to discharging or vice versa.
2	Battery Error: When this bit is set then the <i>PDUSB Device</i> Shall generate a wake event if the Battery has detected an error condition.
15:3	Reserved and Shall Not be used

The *SPM* **May** Enable or Disable the wake events associated with one or more of the above events by using this feature.

If the *PDUSB Hub* is not configured, the *PDUSB Hub*'s response to this request is undefined.

9.4.2.2 CHARGING_POLICY Feature Selector

When the feature selector is set to CHARGING_POLICY, the wIndex field **Shall** be set to one of the values defined in [Table 9.13, "Charging Policy Encoding"](#). If the device is using *USB Type-C* Current above the default value or is using PD then this feature setting has no effect and the rules for power levels specified in the [\[USB Type-C 2.4\]](#) or USB PD specifications **Shall** apply.

Table 9.13 Charging Policy Encoding

Value	Description
00H	The device Shall follow the default current limits as defined in the USB 2.0 or USB 3.1 specification, or as negotiated through other USB mechanisms such as BC. This is the default value.
01H	The Device May draw additional power during the unconfigured and suspend states for the purposes of charging. For charging the device itself, the device Shall limit its current draw to the higher of these two values: ICCHPF as defined in the USB 2.0 or USB 3.1 specification, regardless of its USB state. Current limit as negotiated through other USB mechanisms such as BC.
02H	The Device May draw additional power during the unconfigured and suspend states for the purposes of charging. For charging the device itself, the device Shall limit its current draw to the higher of these two values: ICCLPF as defined in the USB 2.0 or USB 3.1 specification, regardless of its USB state. Current limit as negotiated through other USB mechanisms such as BC.
03H	The device Shall Not consume any current for charging the device itself regardless of its USB state.
04H-FFFFH	Reserved and Shall Not be used

This is a **Valid Command** for the *PDUSB Hub/Peripheral* in the Address or Configured USB states. Further, it is only **Valid** if the device reports a USB PD capability descriptor in its BOS descriptor and Bit 5 of the bmAttributes in that descriptor is set to 1. The device will go back to the wIndex default value of 0 whenever it is reset.

10 Power Rules

10.1 Introduction

The flexibility of power provision on *USB Type-C®* is expected to lead to power adapter re-use and the increasingly widespread provision of USB power outlets in domestic and public places and in transport of all kinds. Environmental considerations could result in unbundled power adapters. Rules are needed to avoid incompatibility between the *Sources* and the *Sinks* they are used to power, in order to avoid user confusion and to meet user expectations. This section specifies a set of rules that *Sources* and *Sinks* **shall** follow. These rules provide a simple and consistent user experience.

The *PDP Rating* is a manufacturer declared value placed on packaging to help the user understand the capabilities of a *Charger* or the size of *Charger* required to power their device. For *PDP* values of 10W and above the *PDP* **shall** be declared as an integer number of Watts. For *PDP* values less than 10W, the *PDP* **shall** be declared in increments of 0.5W.

The *Source Power Rules* define a *PDP* to provide a simple way to tell the user about the capabilities of their power adapter or device. *PDP Rating* is akin to the wattage rating of a light bulb - bigger numbers mean more capability.

The *Sink Power Rules* define a *PDP* to provide a simple way to tell the user which *Sources* will provide adequate power for their *Sink*.

10.2 Source Power Rules

The *Source Power Rules* defined in this section include both **Normative** and **Optional** rules. For all of the defined rules, the capabilities a *Source* exposes are based on the **Port Maximum PDP**, or if power constrained, the **Port Present PDP** of the *Port*.

For a *Guaranteed Capability Port*, the *Source* **shall** always include in every **Source_Capabilities** or **EPR_Source_Capabilities** Message sent to a *Sink* all the (A)PDOs that are defined by the **Normative** (and **Optional** when implemented) rules based on the *Port's* **Port Maximum PDP** and *Mode* of operation (i.e., *SPR Mode* or *EPR Mode*).

For a *Managed Capability Port*, except before the *First Explicit Contract* or before the *Explicit Contract* after the **Port Present PDP** changes on a *Shared Capacity Charger Port*, the *Source* **shall** always include in every **Source_Capabilities** or **EPR_Source_Capabilities** Message sent to a *Sink* all the (A)PDOs that are defined by the **Normative** (and **Optional** when implemented) rules based on the *Port's* **Port Present PDP** and *Mode* of operation (i.e., *SPR Mode* or *EPR Mode*). After the *First Explicit Contract*, this requirement assures that the attached *Sink* will always know what voltages (or voltage modes) are presently available from the *Source*.

In order to meet the expectations of the user, the Maximum Current/Power in the *Source Capabilities* PDO or APDO for *Sources* with a *PDP Rating* of x Watts **shall** be as follows:

- Maximum current for **Normative** and **Optional** Fixed Supply/Variable Supply PDOs **shall** be either RoundUp(x/voltage) or RoundDown(x/voltage) to the nearest 10mA.
- Maximum current for SPR Programmable Power Supply APDOs **shall** be as defined in [Table 10.10, "SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP"](#).

Note: When the Constant Power bit is set in the APDO, the programmable power supply's output current is as defined in [Table 10.10, "SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP"](#) however the programmable power supply will limit its output current so that the product of its actual output voltage times the output current does not exceed the *PDP*.

- If a 9V Prog, 15V Prog or 20V Prog Programmable Power Supply APDO is *Advertised* when not required by [Table 10.10, "SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP"](#), then the maximum current **shall** be RoundDown (x/Prog Voltage) to the nearest 50mA. When the PPS Power Limited bit is clear the *Source* **shall** provide this current at **Maximum Voltage**.
- Maximum power for **Optional** Battery Supply PDOs **shall** be $\leq x$.

10.2.1 Source Power Rule Considerations

The *Source Power Rules* are designed to:

- Ensure the *PDP Rating (PDP)* of an adapter specified in watts explicitly defines the voltages and currents at each voltage the adapter supports.
- Ensure that adapters with a large *PDP Ratings* are always capable of providing the power to devices designed for use with adapters with a smaller *PDP Rating*.
- Enable an ecosystem of adapters that are inter-operable with the devices in the ecosystem.

The considerations that lead to the *Source Power Rules* are based are summarized in [Table 10.1, "Considerations for Sources"](#).

Table 10.1 Considerations for Sources

Considerations	Rationale	Consequence
Simple to identify capability	A user going into an electronics retailer knows what they need	Cannot have a complex identification scheme
Higher power <i>Sources</i> are a superset of smaller ones	Bigger is always better in user's eyes – don't want a degradation in performance	Higher power <i>Sources</i> do everything smaller ones do
Unambiguous <i>Source</i> definitions	<i>Sources</i> with the same power rating but different VI combinations might not inter-operate	To avoid user confusion, any given power rating has a single definition
A range of power ratings	Users and companies will want freedom to pick appropriate <i>Source</i> ratings	Fixed profiles at specific power levels don't provide adequate flexibility, e.g., profiles as defined in previous versions of PD.
5V@3A <i>USB Type-C Source</i> is defined by [USB Type-C 2.4]	5V@3A <i>USB Type-C Source</i> is considered	All > 15W adapters must support 5V@3A or superset consideration is violated
Maximize 3A cable utilization	3A cables will be ubiquitous	Increase to maximum voltage (20V) before increasing current beyond 3A
Optimize voltage rail count	More rails are a higher burden for <i>Sources</i> , particularly in terms of testing	5V is a basic USB requirement. 48V provides the maximum capability.
Some <i>Sources</i> are not able to provide significant power	Some small <i>Battery</i> -operated <i>Sources</i> e.g., mobile devices, are able to provide more power directly from their <i>Battery</i> than from a regulated 5V supply	In addition to the minimal 5V <i>Advertisements</i> are able to <i>Advertise</i> more power from their <i>Battery</i>
Some <i>Sources</i> share power between multiple Ports (<i>Hubs</i> and multi- <i>Port Chargers</i>)	<i>Hubs</i> and multi-port <i>Chargers</i> have to be supported	See Section 10.3, "Sink Power Rules"

10.2.2 Normative Voltages and Currents

The voltages and currents an *SPR Source* with a *PDP Rating* of x Watts **shall** support are as defined in [Table 10.2, "SPR Normative Voltages and Minimum Currents"](#).

Table 10.2 SPR Normative Voltages and Minimum Currents

Port Maximum PDP Rating (W)	5V Fixed	9V Fixed	15V Fixed	20V Fixed	SPR AVS
$0.5 \leq x \leq 15$	$(PDP/5)A^3$	-	-	-	-
$15 < x \leq 27$	$3A^2$	$(PDP/9)A^3$	-	-	-
$27 < x \leq 45$	$3A^2$	$3A^2$	$(PDP/15)A^3$	-	(9V – 15V): • (15V Fixed Supply Max Current) A
$45 < x \leq 60$	$3A^2$	$3A^2$	$3A^2$	$(PDP/20)A^3$	(9V – 15V): • (15V Fixed Supply Max Current) A ⁴ (15V – 20V): • (20V Fixed Supply Max Current) A
$60 < x \leq 100$	$3A^2$	$3A^2$	$3A^2$	$(PDP/20)A^{1,3}$	(9V – 15V): • (15V Fixed Supply Max Current) A ^{4,5} (15V – 20V): • (20V Fixed Supply Max Current) A ^{1,5}
1)	Requires a 5A cable.				
2)	The <i>Fixed Supply</i> PDOs Maximum Current field shall Advertise at least 3A, but may Advertise up to RoundUp (<i>PDP/voltage</i>) to the nearest 10mA. Requires a 5A cable if over 3A is Advertised.				
3)	The <i>Fixed Supply</i> PDOs Maximum Current field shall Advertise either RoundDown (<i>PDP/voltage</i>) or RoundUp (<i>PDP/voltage</i>) to the nearest 10mA.				
4)	<i>SPR AVS</i> current for this voltage range is the maximum current as Advertised by the 15V Fixed Supply PDO. This current can be higher than 3A (refer to Note 2). Requires a 5A cable if over 3A is Advertised.				
5)	The <i>Sink</i> is allowed to request up to the 20V Fixed Supply Max Current when the requested voltage is 15.0V.				

SPR *Managed Capability Ports* when power constrained are defined to offer **Valid (A)PDOs** based on the port's **Port Maximum PDP** (as per [Table 10.2, "SPR Normative Voltages and Minimum Currents"](#)) at lower **Port Present PDP** (as per [Table 10.3, "SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP"](#)) because these voltages would otherwise be available if the *Managed Capability Port* power hadn't been constrained.

Table 10.3 SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP

Port Present PDP (W)	5V Fixed	9V Fixed	15V Fixed	20V Fixed	SPR AVS with Max Voltage of 15V or 20V per Table 10.2 ⁶
$5 < x \leq 15$	$(PDP/5)A^3$	$(PDP/9)A^{3,7,8}$	$(PDP/15)A^{3,7,8}$	$(PDP/20)A^{3,7,8}$	(9V – 15V): <ul style="list-style-type: none"> (15V Fixed Supply Max Current) $A^{4,6,8}$ (15V – 20V): <ul style="list-style-type: none"> (20V Fixed Supply Max Current) $A^{6,8}$
$15 < x \leq 27$	$3A^2$	$(PDP/9)A^3$	$(PDP/15)A^{3,7}$	$(PDP/20)A^{3,7}$	(9V – 15V): <ul style="list-style-type: none"> (15V Fixed Supply Max Current) A^4 (15V – 20V): <ul style="list-style-type: none"> (20V Fixed Supply Max Current) A^6
$27 < x \leq 45$	$3A^2$	$3A^2$	$(PDP/15)A^3$		
$45 < x \leq 60$	$3A^2$	$3A^2$	$3A^2$	$(PDP/20)A^3$	(9V – 15V): <ul style="list-style-type: none"> (15V Fixed Supply Max Current) A^4 (15V – 20V): <ul style="list-style-type: none"> (20V Fixed Supply Max Current) A
$60 < x \leq 100$	$3A^2$	$3A^2$	$3A^2$	$(PDP/20)A^{1,3}$	(9V – 15V): <ul style="list-style-type: none"> (15V Fixed Supply Max Current) $A^{4,5}$ (15V – 20V): <ul style="list-style-type: none"> (20V Fixed Supply Max Current) $A^{1,5}$
1) Requires a 5A cable. 2) The <i>Fixed Supply</i> PDOs Maximum Current field Shall Advertise at least 3A, but May Advertise up to RoundUp ($PDP/voltage$) to the nearest 10mA. Requires a 5A cable if over 3A is Advertised. 3) The <i>Fixed Supply</i> PDOs Maximum Current field Shall Advertise either RoundDown ($PDP/voltage$) or RoundUp ($PDP/voltage$) to the nearest 10mA. 4) SPR AVS current for this voltage range is the maximum current as Advertised by the 15V Fixed Supply PDO. This current can be higher than 3A (refer to Note 2). Requires a 5A cable if over 3A is Advertised. 5) The Sink is allowed to request up to the 20V Fixed Supply Max Current when the requested voltage is 15.0V. 6) The Max Voltage for SPR AVS is what is allowed by Table 10.2, "SPR Normative Voltages and Minimum Currents" based on the port's Port Maximum PDP . 7) This SPR Fixed Supply voltage is only available if allowed by Table 10.2, "SPR Normative Voltages and Minimum Currents" based on the port's Port Maximum PDP . 8) SPR Sources May offer (A)PDOs at this Port Present PDP					

In reference to [Table 10.3, "SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP"](#), [Table 10.4, "SPR Source Port Present PDP less than Port Maximum PDP Examples"](#) gives examples of which SPR capabilities are *Advertised* based on **Port Present PDP** on a *Managed Capability Port* and the port's **Port Maximum PDP** and cable's current rating.

Table 10.4 SPR Source Port Present PDP less than Port Maximum PDP Examples

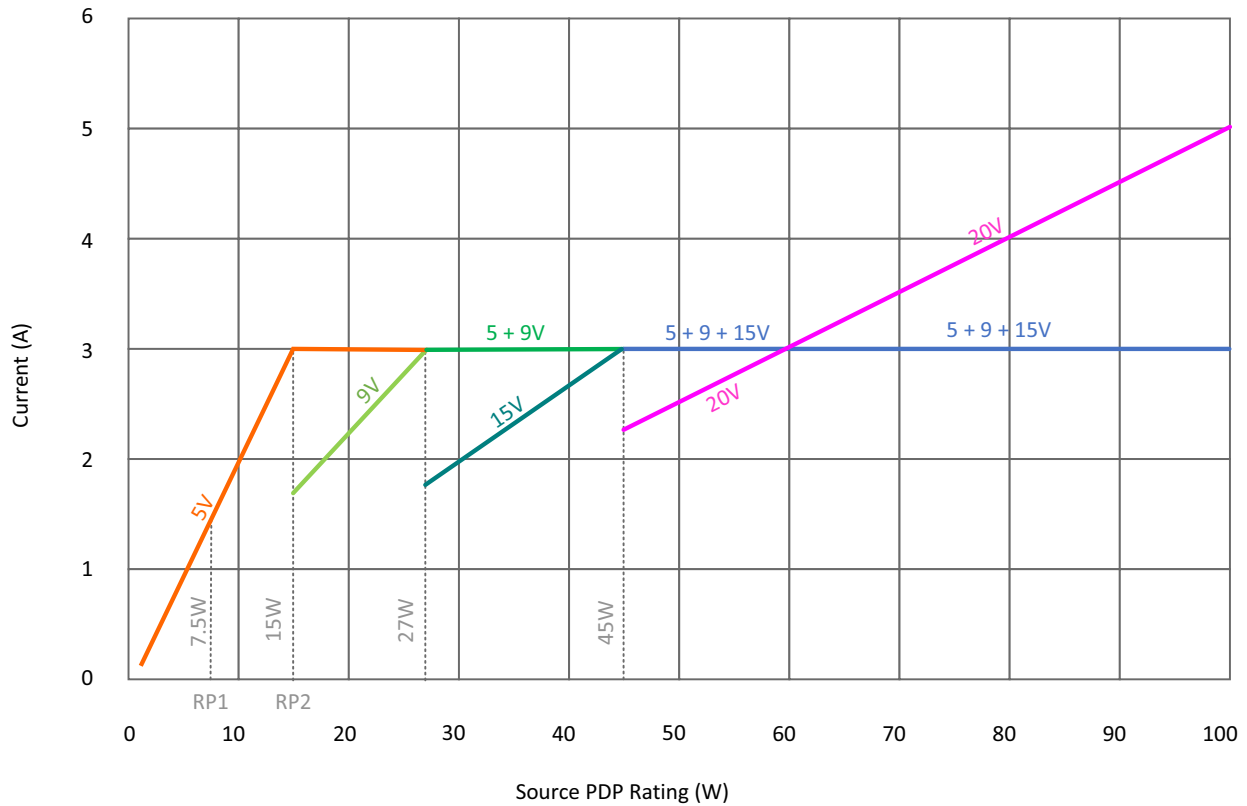
Port Maximum PDP and Cable Rating	Port Present PDP	Offers				
		5V Fixed	9V Fixed	15V Fixed	20V Fixed	SPR AVS
80W / 5A	65W	3A ¹	3A ¹	3A ¹	3.25A	9V – 15V: 3A 15V – 20V: 3.25A
80W / 5A	40W	3A ¹	3A ¹	2.67A	2A	9V – 15V: 2.67A 15V – 20V: 2A
80W / 3A	40W	3A ¹	3A	2.67A	2A	9V – 15V: 2.67A 15V – 20V: 2A
40W / 5A	40W	3A ¹	3A ¹	2.67A	Not Offered	9V – 15V: 2.67A
40W / 3A	40W	3A ¹	3A	2.67A	Not Offered	9V – 15V: 2.67A
80W / 5A	20W	3A ¹	2.22A	1.33A	1A	9V – 15V: 1.33A 15V – 20V: 1A
80W / 3A	20W	3A ¹	2.22A	1.33A	1A	9V – 15V: 1.33A 15V – 20V: 1A
40W / 5A	20W	3A ¹	2.22A	1.33A	Not Offered	9V – 15V: 1.33A
40W / 3A	20W	3A ¹	2.22A	1.33A	Not Offered	9V – 15V: 1.33A
80W/3A	15W	3A	1.67A ²	1A ²	0.75A ²	9V - 15V: 1A ² 15V - 20V: 0.75A ²
40W/3A	15W	3A	1.67A ²	1A ²	Not offered	9V - 15V: 1A ²
1)	The <i>Fixed Supply</i> PDO Maximum Current field will <i>Advertise</i> at least 3A but May <i>Advertise</i> up to RoundUp (<i>PDP</i> /voltage) to the nearest 10mA.					
2)	These <i>Capabilities</i> are not required but may be offered at this Port Present PDP .					

10.2.2.1 Fixed Supply PDOs

[Figure 10.1, "SPR Source Power Rule Illustration for Fixed Supply PDOs"](#) illustrates the minimum current that an SPR Source **Shall** support at each voltage for a given PDP Rating for Fixed Supply PDOs.

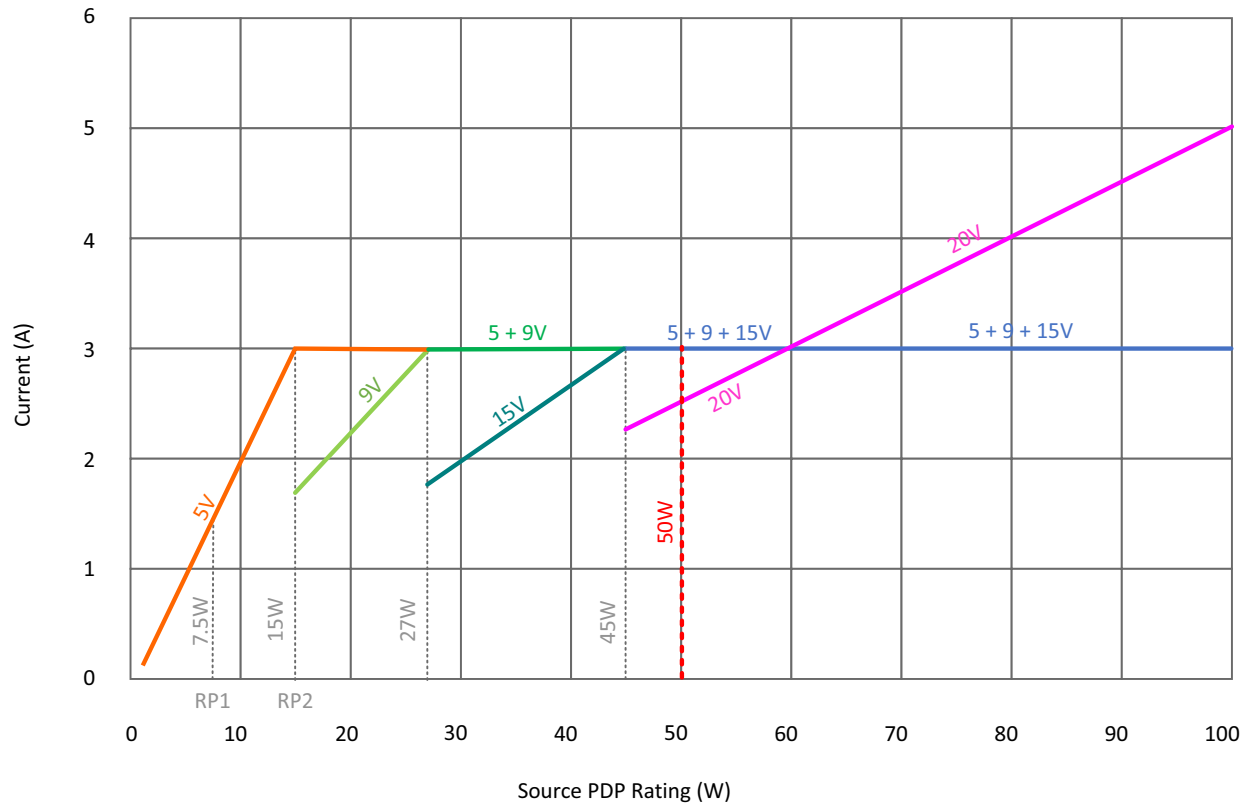
Note: Not illustrated are that currents higher than 3A are allowed to be offered up to a limit of 5A given that a 5A cable is detected by the Source and the voltage times current remains within the Source PDP Rating.

Figure 10.1 SPR Source Power Rule Illustration for Fixed Supply PDOs



[Figure 10.2, "SPR Source Power Rule Example For Fixed Supply PDOs"](#) shows an example of an adapter with a rating at 50W. The adapter is required to support 20V at 2.5A, 15V at 3A, 9V at 3A and 5V at 3A.

Figure 10.2 SPR Source Power Rule Example For Fixed Supply PDOs



[Table 10.5, "Fixed Supply PDO - Source 5V"](#), [Table 10.6, "Fixed Supply PDO - Source 9V"](#), [Table 10.7, "Fixed Supply PDO - Source 15V"](#) and [Table 10.8, "Fixed Supply PDO - Source 20V"](#) show the *Fixed Supply PDOs* that **shall** be supported for each of the **Normative** voltages defined in [Table 10.2, "SPR Normative Voltages and Minimum Currents"](#).

Table 10.5 Fixed Supply PDO - Source 5V

Bit(s)	Description	
B31...30	<i>Fixed Supply</i>	
B29	<i>Dual-Role Power</i>	
B28	<i>USB Suspend Supported</i>	
B27	<i>Unconstrained Power</i>	
B26	<i>USB Communications Capable</i>	
B25	<i>Dual-Role Data</i>	
B24	<i>Unchunked Extended Messages Supported</i>	
B23	<i>EPR Capable</i>	
B22	Reserved – <i>Shall</i> be set to zero.	
B21...20	<i>Peak Current</i>	
B19...10	5V	
B9...0	Current based on <i>PDP</i>	
	PDP Rating (x)	Current (A)
	$0.5 \leq x \leq 15$	$x \div 5$
	$15 < x \leq 25$	$3 \leq A \leq x \div 5$
	$25 < x \leq 100$	$3 \leq A \leq 5$

Table 10.6 Fixed Supply PDO - Source 9V

Bit(s)	Description	
B31...30	<i>Fixed Supply</i>	
B29...22	Reserved – <i>Shall</i> be set to zero.	
B21...20	Peak Current	
B19...10	9V	
B9...0	Current based on <i>PDP</i>	
	PDP Rating (x)	Current (A)
	$0.5 \leq x \leq 15$	PDO not required
	$15 < x \leq 27$	$x \div 9$
	$27 < x \leq 45$	$3 \leq A \leq x \div 9$
	$45 < x \leq 100$	$3 \leq A \leq 5$

Table 10.7 Fixed Supply PDO - Source 15V

Bit(s)	Description	
B31...30	<i>Fixed Supply</i>	
B29...22	Reserved – <i>Shall</i> be set to zero.	
B21...20	Peak Current	
B19...10	15V	
B9...0	Current based on <i>PDP</i>	
	PDP Rating (x)	Current (A)
	$0.5 \leq x \leq 27$	PDO not required
	$27 < x \leq 45$	$x \div 15$
	$45 < x \leq 75$	$3 \leq A \leq x \div 15$
	$75 < x \leq 100$	$3 \leq A \leq 5$

Table 10.8 Fixed Supply PDO - Source 20V

Bit(s)	Description	
B31...30	<i>Fixed Supply</i>	
B29...22	Reserved – <i>Shall</i> be set to zero.	
B21...20	Peak Current	
B19...10	20V	
B9...0	Current based on <i>PDP</i>	
	PDP Rating (x)	Current (A)
	$0.5 \leq x \leq 45$	PDO not required
	$45 < x \leq 100$	$x \div 20$

More current **May** be offered in the PDOs when **Optional** voltages/currents are supported and a 5A cable is being used (see [Section 10.2.3, "Optional Voltages/Currents"](#)).

10.2.2.2 SPR Adjustable Voltage Supply (AVS)

For SPR AVS, [Figure 10.3, "Valid SPR AVS Operating Region for a Source advertising in the range of \$27W < PDP \leq 45W\$ "](#), [Figure 10.4, "Valid SPR AVS Operating Region for a Source advertising in the range of \$45W < PDP \leq 60W\$ "](#) and [Figure 10.5, "Valid SPR AVS Operating Region for a Source advertising in the range of \$60W < PDP \leq 100W\$ "](#) illustrate the valid operating region for SPR AVS RDO requests in the ranges of $27W < PDP \leq 45W$, $45W < PDP \leq 60W$ and $60W < PDP \leq 100W$, respectively.

Figure 10.3 Valid SPR AVS Operating Region for a Source advertising in the range of $27W < PDP \leq 45W$

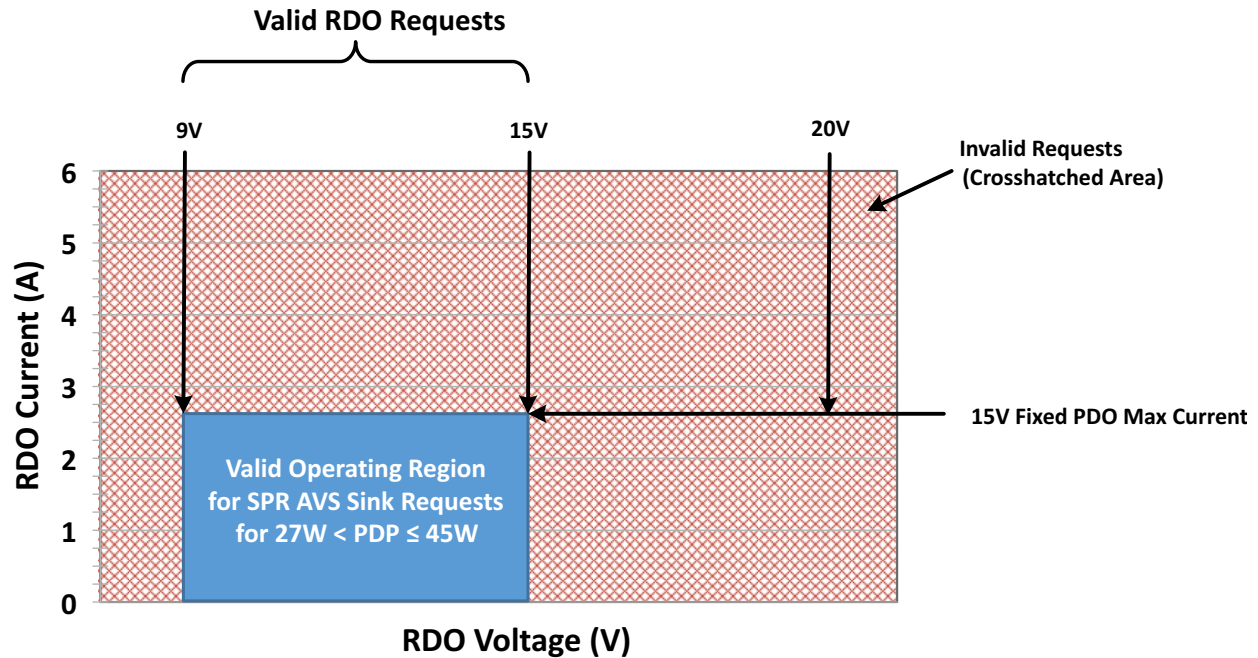


Figure 10.4 Valid SPR AVS Operating Region for a Source advertising in the range of $45W < PDP \leq 60W$

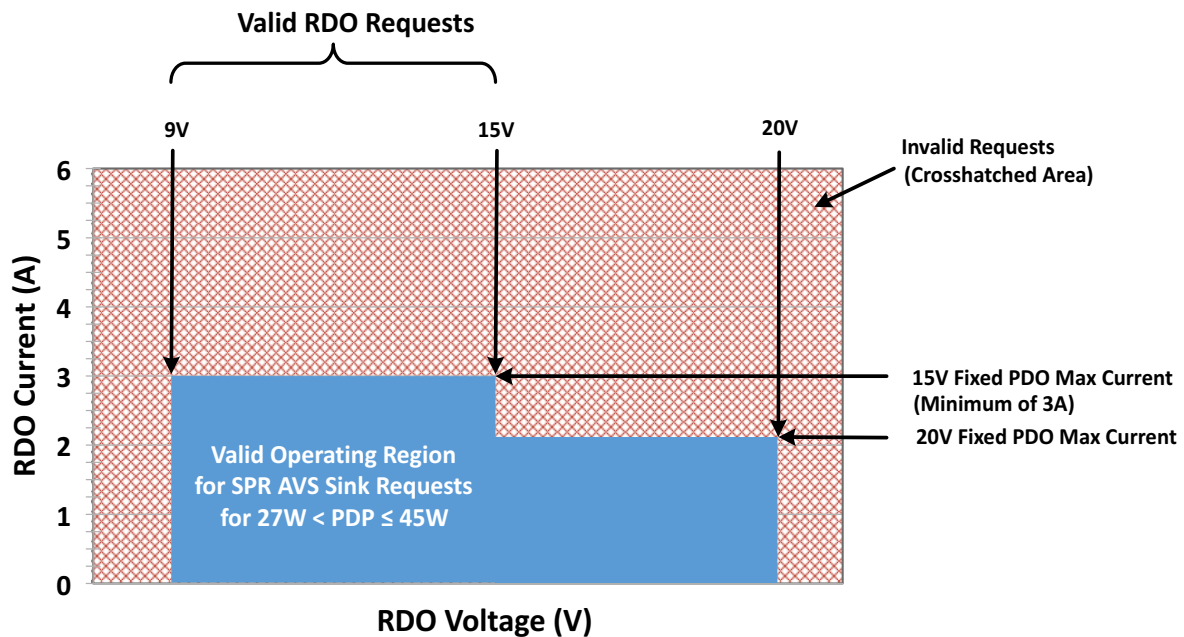
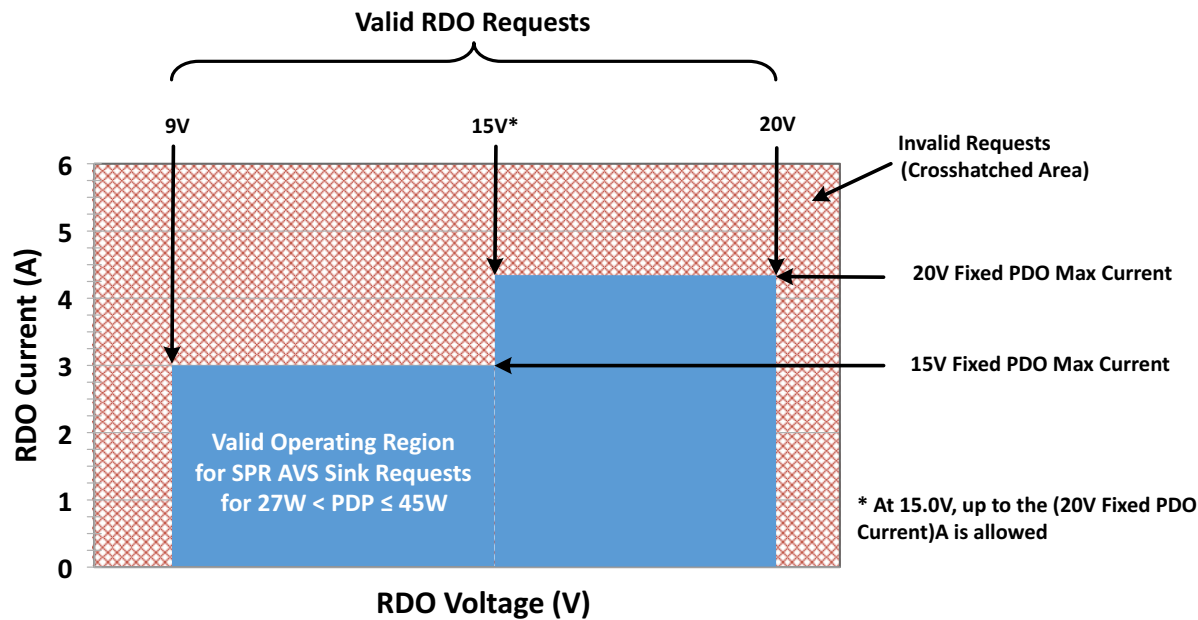


Figure 10.5 Valid SPR AVS Operating Region for a Source advertising in the range of $60W < PDP \leq 100W$



10.2.2.2.1 SPR Adjustable Voltage Supply (AVS) Voltage Ranges

Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable" shows the Minimum and Maximum Voltage for the SPR AVS ranges.

Table 10.9 SPR Adjustable Voltage Supply (AVS) Voltage Ranges

	AVS Voltage Range	
	15V AVS	20V AVS
Maximum Voltage	15V	20V
Minimum Voltage	9V	9V

The voltage output at the Source's connector **shall** be +/-5% for both the Maximum Voltage and the Minimum Voltage.

10.2.3 Optional Voltages/Currents

10.2.3.1 Optional Normative Fixed, Variable and Battery Supply

In addition to the voltages and currents specified in [Section 10.2.2, "Normative Voltages and Currents"](#), an *SPR Source* that is optimized for use with a specific *Sink* or a specific class of *Sinks* **May Optionally** supply additional voltages and increased currents. However, the **Optional** voltages **Shall Not** exceed 9V.

Optional voltages **Shall Not** be implemented on *EPR Source* including for both *SPR Mode* and *EPR Modes* of operation. *EPR* versions of *Variable Supply* and *Battery Supply* *PDOs* are not defined and **Shall Not** be implemented, however *SPR Variable Supply* and *Battery Supply* *PDOs* are allowed in *EPR Mode*.

While allowed, the use of **Optional** voltages and currents is not recommended as two *Sources* with the same *PDP Rating* but not supporting the same **Optional** voltages and currents can behave differently thus confusing the user.

See [Section 10.2, "Source Power Rules"](#) for the rules that **Shall** apply to **Optional** *PDOs* in order to be consistent with the declared *PDP Rating* and the **Normative** voltages and currents.

10.2.3.2 Optional Normative SPR Programmable Power Supply

The voltages and currents a Programmable Power Supply with a *PDP Rating* of x Watts **Shall** support are as defined [Table 10.10, "SPR Programmable Power Supply *PDOs* and *APDOs* based on the Port Maximum *PDP*"](#).

When **Optional** Programmable Power Supply *APDOs* are offered, the following requirements **Shall** apply:

- A *Source* that Advertises **Optional** Programmable Power Supply *APDOs* **Shall** Advertise the *PDOs* and *APDOs* shown in [Table 10.10, "SPR Programmable Power Supply *PDOs* and *APDOs* based on the Port Maximum *PDP*"](#).
- A *Source* **Shall** Advertise **Optional** Programmable Power Supply *APDOs* with Maximum Voltage and Minimum Voltages for nominal voltage as defined in [Table 10.11, "SPR Programmable Power Supply Voltage Ranges"](#).
- A *Source* **Shall Not** Advertise a Programmable Power Supply *APDO* that does not follow the Minimum Voltage and Maximum Voltage defined in [Table 10.11, "SPR Programmable Power Supply Voltage Ranges"](#).
- In no case **Shall** a *Source* Advertise a current that exceeds the *Attached* cable's current rating.
- The Max Voltage **Shall Not** exceed 21V while in *SPR Mode*.

Table 10.10 SPR Programmable Power Supply *PDOs* and *APDOs* based on the Port Maximum *PDP*

PDP Maximum PDP (W)	SPR Fixed and AVS	9V Prog ³	15V Prog ³	20V Prog ³
x < 15W	Required per Table 10.2, "SPR Normative Voltages and Minimum Currents" (or Table 10.3, "SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP" when applicable)	-	-	-
15W		-	-	-
15 < x < 27W		(PDP/9)A ¹	-	-
27W		3A	-	-
27 < x < 45W		3A ²	(PDP/15)A ¹	-
45W		-	3A	-
45 < x < 60W		-	3A ²	(PDP/20)A ¹
60W		-	-	3A
60 < x < 100W		-	-	(PDP/20)A ¹
100W		-	-	5A
1)	The <i>SPR PPS APDOs</i> Maximum Current field Shall Advertise RoundDown (PDP/Prog Voltage) to the nearest 50mA.			
2)	The <i>SPR PPS APDOs</i> Maximum Current field Shall Advertise at least 3A, but May Advertise up to RoundDown(PDP/Prog Voltage) to the nearest 50mA.			
3)	Applies to <i>APDOs</i> regardless of value of the PPS Power Limited bit.			

10.2.3.2.1 SPR Programmable Power Supply Voltage Ranges

The *SPR PPS* Voltage ranges map to the *Fixed Supply* Voltages. For each fixed voltage there is a defined voltage range for the matching *SPR PPS APDO*. [Table 10.11, "SPR Programmable Power Supply Voltage Ranges"](#) shows the Minimum and Maximum Voltage for the Programmable Power Supply that corresponds to the Fixed nominal voltage.

Table 10.11 SPR Programmable Power Supply Voltage Ranges

	Fixed Nominal Voltage		
	9V Prog	15V Prog	20V Prog
Maximum Voltage	11V	16V	21V
Minimum Voltage	5V	5V	5V

The voltage output at the *Source's* connector **shall** be +/-5% for both the Maximum Voltage and the Minimum Voltage.

10.2.3.2.2 Examples of the use of SPR Programmable Power Supplies

The following examples illustrate what a power adapter that *Advertises* a particular *PDP Rating* **May** offer:

- 1) *PDP 27W* implementation includes:
 - 5V @ 3A,
 - 9V @ 3A, and
 - 9V Prog @ 3A.
- 2) *PDP 36W* implementation includes:
 - 5V @ 3A,
 - 9V @ 3A,
 - 15 @ 2.4A,
 - *SPR AVS* with 9V - 15V @ 2.4A,
 - 9V Prog @ 3 A, and
 - 15V Prog @ 2.4A.
- 3) *PDP 36W* implementation that **Optionally** includes higher current in the 9V Prog PPS:
 - 5V @ 3A,
 - 9V @ 3A,
 - 15 @ 2.4A,
 - *SPR AVS* with 9V - 15V @ 2.4A,
 - 9V Prog @ >3A up to 4A (with a 5A cable) and 15V
 - Prog @ 2.4A.
- 4) *PDP 50W* implementation includes:
 - 5V @ 3A,
 - 9V @ 3A,
 - 15 @ 3A,
 - 20V @ 2.5A,

- *SPR AVS* with 9V - 15V @ 3A & 15V - 20V @ 2.5A,
 - 15V Prog @ 3A, and
 - 20V Prog @ 2.5A.
- 5) *PDP* 80W implementation includes:
- 5V @ 3A,
 - 9V @ 3A,
 - 15 @ 3A,
 - 20V @ 4A,
 - *SPR AVS* with 9V - 15V @ 3A & 15V - 20V @ 4A,
 - 15V Prog @ 3A, and
 - 20V Prog @ 4A.

The first example illustrates a basic example of a supply that can only support 5V and 9V.

The second and third examples illustrates as the *PDP Rating* goes higher there are more possible combinations that meet the *Power Rules*. These examples also add *SPR AVS*. Although there are multiple ways to meet the *Power Rules*, while operating in *SPR Mode* no more than a combination of seven *SPR (A)PDOs* and *APDOs* can be offered.

The fourth and fifth example show that the 15V Prog @ 3A fully covers the 9V Prog @3A range so it is not necessary to *Advertise* both. These examples also illustrate *SPR AVS* being extended up to 20V with separate current limits for the 9V - 15V and 15V - 20V ranges - a single *SPR AVS APDO* covers advertising both ranges.

10.2.3.3 Optional Normative Extended Power Range (EPR)

Support of *EPR Mode* is **Optional**. An *EPR Capable* port has a *PDP Rating* that is $>100\text{W}$ and $\leq 240\text{W}$. An *EPR Capable Source Port (EPR Source Port)* **May** operate in either *SPR Mode* or *EPR Mode* when operating at 100W or less.

An *EPR Source Port* operating in *SPR Mode* **May** offer less than 100W to avoid violating safety regulations. When operating in *EPR Mode*, an *EPR Source Port* **Shall** offer 100W in Fixed 20V when not constrained by multi-port sharing limits.

An *EPR Source* **May** include multiple ports and these ports can be functionally implemented as *Shared Capacity Charger* or *Assured Capacity Charger* ports as defined in [\[USB Type-C 2.4\]](#).

Any port on an *EPR Source* that has a **Port Present PDP** of 100W or less **Shall** follow the **Normative** requirements for *SPR Source Ports* and **Shall** operate only in *SPR Mode*. Any port on an *EPR Source* that is operating with a cable that is not *EPR Capable* **Shall** operate only in *SPR Mode*. An *EPR Source*, when operating in *SPR Mode* with a 5A cable, **May** offer less than 5A due to design tolerances in order to meet applicable safety standards. For best user experience it **Should** be as close to 100W as possible.

[Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable"](#) and [Table 10.13, "EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable"](#) define the **Normative** requirements *EPR Source Ports*. While not included in these tables, any *EPR Source Port* that also supports *SPR PPS* **Shall** offer the *SPR Fixed 20V PDO* and *PPS 20V Prog APDO* at 100W (or the maximum available *PDP* when the port is operating at an Equivalent *PDP* $<100\text{W}$) when in *EPR Mode*:

- When an *EPR Source Port* is capable of supplying its *PDP Rating*, it **Shall** adhere to the requirements defined in [Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable"](#) based on its *PDP Rating* of x Watts.
- When a *Source Port* on an *EPR Charger* is unable to provide its **Port Maximum PDP**, it **Shall** adhere to the requirements defined in [Table 10.13, "EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable"](#) based on a **Port Present PDP** of x Watts. Some examples:
 - An *EPR Source Port* **May** be unable to provide its *PDP Rating* because it is thermally constrained at the time of power *Negotiation*.
 - A Shared port on a multi-port *EPR Charger* that is limited by the remaining available power.
- When an *EPR Charger* is in an *Adjustable Voltage Supply (AVS) Explicit Contract*:
 - It **Shall** Reject all Requests outside of the defined voltage range (see [Table 10.15, "EPR Adjustable Voltage Supply \(AVS\) Voltage Ranges"](#)) or for a requested voltage and Current that results in a power level that is more than the Port's *Advertised PDP*.
 - In no case **Shall** a *Source* *Advertise* a Current or accept a Current requested by a *Sink* that exceeds the *Attached cable's* current rating.

- The Max Voltage offered by an *EPR Source* **Shall Not** exceed 48V.

Table 10.12 EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable

Port Maximum PDP (W)	SPR Fixed and AVS	28V Fixed	36V Fixed ³	48V Fixed	EPR AVS ^{3, 4}
100 < x ≤ 140	Required per Table 10.2, "SPR Normative Voltages and Minimum Currents" (or Table 10.3, "SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP" when applicable)	(PDP/28)A ²	N/A ¹	N/A ¹	(15V – PDP/5A): • 5A (>PDP/5A – 28V): • (PDP/AVS voltage) A
140 < x ≤ 180		5A	(PDP/36)A ²	N/A ¹	(15V – PDP/5A): • 5A (>PDP/5A – 36V): • (PDP/AVS voltage) A
180 < x ≤ 240		5A	5A	(PDP/48)A ²	(15V – PDP/5A): • 5A (>PDP/5A – 48V): • (PDP/AVS voltage) A
1) <i>EPR SourceS</i> are disallowed from offering <i>Fixed Supply</i> voltages that are above the defined voltages for a given <i>PDP</i> , e.g., 36V is disallowed for any <i>PDP</i> of 140W or lower.					
2) The Fixed PDOs Maximum Current field Shall Advertise either RoundDown (<i>PDP</i> /voltage) or RoundUp (<i>PDP</i> /voltage) to the nearest 10mA.					
3) <i>EPR SourceS</i> Shall reject any request for more than the <i>Advertised PDP</i> , i.e., when output voltage and operating current requested in the <i>Sink RDO</i> is outside of the defined <i>AVS</i> voltage and current range represented by the <i>Advertised PDP</i> , the <i>RDO</i> will be rejected.					
4) The current available for a given <i>AVS</i> voltage is as indicated in this column. The current defined here is describing the top edge of the Valid Operating Region as illustrated in Figure 10.6, "Valid EPR AVS Operating Region" . The <i>AVS APDO</i> does not have a Maximum Current field, so the maximum current has to be calculated from the <i>PDP</i> .					

Table 10.13 EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable

Port Present PDP (W)	SPR Fixed and AVS	28V Fixed	36V Fixed ⁴	48V Fixed ⁴	EPR AVS with Max Voltage of 28V, 36V or 48V per Table 10.12 ^{2, 5, 6}		
					28V	36V	48V
7.5 ≤ x ≤ 15	Required per Table 10.2, "SPR Normative Voltages and Minimum Currents" (or Table 10.3, "SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP" when applicable)	(PDP/28) A ³	(PDP/36) A ³	(PDP/48) A ³	<ul style="list-style-type: none">(15V – max voltage):<ul style="list-style-type: none">(PDP/AVS Voltage) A³		
15 < x ≤ 27		(PDP/28) A ¹					
27 < x ≤ 45							
45 < x ≤ 60							
60 < x ≤ 100							
100 < x ≤ 140	Table 10.3, "SPR Source Capabilities When Port Present PDP is less than Port Maximum PDP" with a Port Present PDP of 100W.	5A	5A	(PDP/48) A ¹	Up to 75W: <ul style="list-style-type: none">(15V – max voltage):<ul style="list-style-type: none">(PDP/AVS voltage) A Above 75W: <ul style="list-style-type: none">(15V – PDP/5A):<ul style="list-style-type: none">5A(>PDP/5A – max voltage):<ul style="list-style-type: none">(PDP/AVS voltage) A		
140 < x ≤ 180		5A					
180 < x ≤ 240		5A					
<div>1) The <i>Fixed Supply</i> PDOs Maximum Current field shall <i>Advertise</i> either RoundDown (PDP/voltage) or RoundUp (PDP/voltage) to the nearest 10mA.</div> <div>2) <i>EPR Sources shall</i> reject any Request for more than the <i>Advertised PDP</i>, i.e., when output voltage and operating current requested in the <i>Sink RDO</i> is outside of the defined <i>AVS voltage</i> and current range represented by the <i>Advertised PDP</i>, the <i>RDO</i> will be rejected.</div> <div>3) <i>EPR Sources may</i> offer an (A)PDOS at this Port Present PDP. When offered, the <i>Fixed Supply</i> PDOs Maximum Current field shall <i>Advertise</i> either RoundDown (PDP/Voltage) or RoundUp (PDP/Voltage) to the nearest 10mA.</div> <div>4) This <i>EPR Fixed Supply</i> voltage is only available if allowed by Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable" based on the port's <i>PDP Rating</i>.</div> <div>5) The Max Voltage for AVS is what is allowed by Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable" based on the port's Port Maximum PDP.</div> <div>6) The current available based on AVS voltage is as indicated in this column. The current defined here is describing the top edge of the Valid Operating Region as illustrated in Figure 10.6, "Valid EPR AVS Operating Region". AVS APDO does not have a Maximum Current field so the maximum current has to be calculated from the <i>PDP</i>.</div>							

Note: *EPR Managed Capability Ports* when power constrained are defined to offer higher voltages at lower **Port Present PDP** (as per [Table 10.13, "EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable"](#)) than the port's **Port Maximum PDP** (as per [Table 10.12, "EPR Source Capabilities based on the Port Maximum PDP and using an EPR Capable Cable"](#)) because these voltages would otherwise be available if the *Managed Capability Port* power hadn't been constrained. *Managed Capability Ports* are required to be properly identified to the user based on the port's **Port Maximum PDP**.

In reference to [Table 10.13, "EPR Source Capabilities when Port Present PDP is less than Port Maximum PDP and using an EPR-capable cable"](#), [Table 10.14, "EPR Source Examples when Port Present PDP is less than Port Maximum PDP"](#) gives examples of which *EPR Capabilities*, in addition to the required *SPR Fixed Supply* PDOs and *SPR AVS APDO*, are *Advertised* based on **Port Present PDP** and the port's **Port Maximum PDP**.

Table 10.14 EPR Source Examples when Port Present PDP is less than Port Maximum PDP

Port Maximum PDP	Port Present PDP	Offers			
		28V Fixed	36V Fixed	48V Fixed	AVS
200W	108W	3.86A	3A	2.25A	48V@108W
160W	108W	3.86A	3A	Not offered	36V@108W
120W	108W	3.86A	Not offered	Not offered	28V@108W
200W	72W	2.57A	2A	1.5A	48V@72W
160W	72W	2.57A	2A	Not offered	36V@72W
120W	72W	2.57A	Not offered	Not offered	28V@72W
200W	36W	1.29A	1A ¹	0.75A ¹	48V@36W ¹
1) These <i>Capabilities</i> are not required but may be offered at this Port Present PDP .					

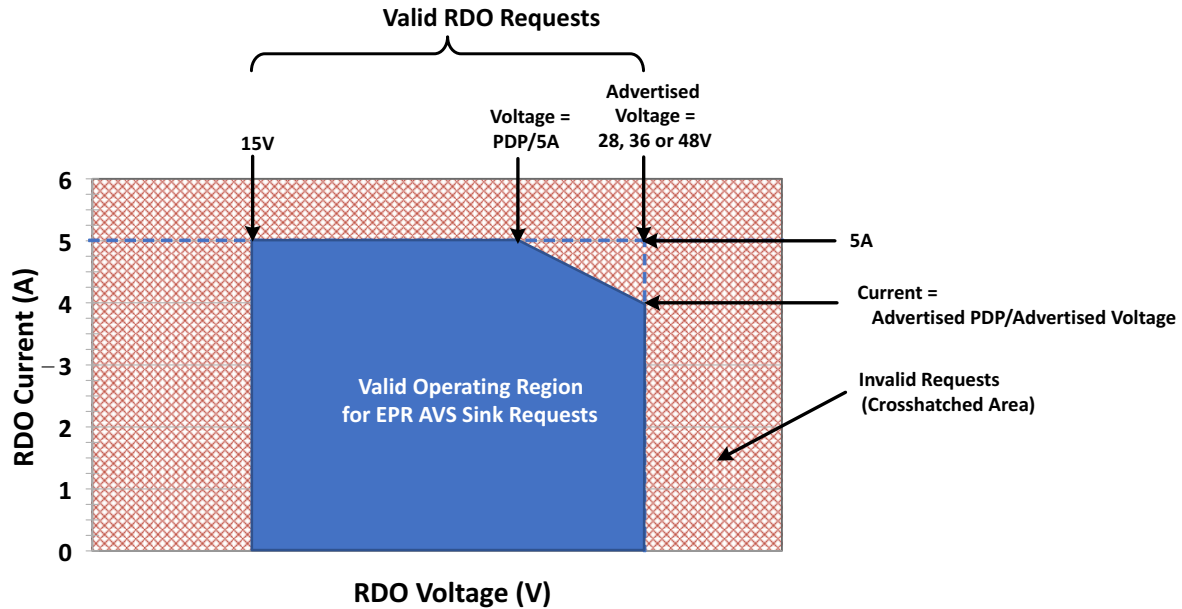
Table 10.14 EPR Source Examples when Port Present PDP is less than Port Maximum PDP

Port Maximum PDP	Port Present PDP	Offers			
		28V Fixed	36V Fixed	48V Fixed	AVS
160W	36W	1.29A	1A ¹	Not offered	36V@36W ¹
120W	36W	1.29A	Not offered	Not offered	28V@36W

1) These *Capabilities* are not required but may be offered at this *Port Present PDP*.

EPR Sources when operating in an AVS Explicit Contract are required to stay within their PDP as such they **Shall** respond to any request (VA) for more than the PDP with a **Reject** Message. [Figure 10.6, "Valid EPR AVS Operating Region"](#) illustrates the definition of the **Valid** operating range for an EPR Source operating in an AVS Explicit Contract based on its *Advertised PDP*.

Figure 10.6 Valid EPR AVS Operating Region



[Figure 10.7, "EPR Source Power Rule Illustration for Fixed PDOs"](#) illustrates the minimum current that an EPR Source **Shall** support at each voltage for a given PDP Rating.

Note: Not illustrated are that currents higher than 3A are allowed to be offered up to a limit of 5A given that a 5A cable is detected by the *Source* and the voltage times current remains within the *Source PDP Rating*.

The graph illustrates the relationship between the Source PDP Rating (W) on the x-axis and the Current (A) on the y-axis. The x-axis ranges from 0 to 240 W, and the y-axis ranges from 0 to 6 A. The graph shows several power profiles and their corresponding current limits:

- RP1:** 7.5W, 15W, 27W, 45W, 100W, 140W, 180W.
- RP2:** 15W, 27W, 45W, 100W, 140W, 180W.
- Power Profiles:**
 - 5V:** 0W to 15W, 0A to 3A.
 - 9V:** 15W to 27W, 1.67A to 3A.
 - 15V:** 27W to 45W, 1.8A to 3A.
 - 20V:** 45W to 100W, 2.25A to 5A.
 - 28V:** 100W to 140W, 3.5A to 5A.
 - 36V:** 140W to 180W, 3.9A to 5A.
 - 48V:** 180W to 240W, 3.8A to 5A.
- Current Limits:**
 - 5 + 9V:** 27W to 45W, 3A.
 - 5 + 9 + 15V:** 45W to 100W, 3A.
 - 5 + 9 + 15V:** 100W to 140W, 3A.
 - 5 + 9 + 15V:** 140W to 180W, 3A.
 - 5 + 9 + 15V:** 180W to 240W, 3A.

Table 10.15, "EPR Adjustable Voltage Supply (AVS) Voltage Ranges" shows the Minimum and Maximum Voltage for the EPR AVS ranges.

	AVS Voltage Ranges		
	28V AVS	36V AVS	48V AVS
Maximum Voltage	28V	36V	48V
Minimum Voltage	15V	15V	15V

Page 1012 Universal Serial Bus Power Delivery Specification, Revision 3.2, Version 1.1, 2024-10

10.3 Sink Power Rules

10.3.1 Sink Power Rule Considerations

The *Sink Power Rules* are designed to ensure the best possible user experience when a given *Sink* used with a compliant *Source* of arbitrary Output Power Rating that only supplies the **Normative** voltages and currents.

The *Sink Power Rules* are based on the following considerations:

- Low power *Sources* (e.g., 5V) are expected to be very common and will be used with *Sinks* designed for a higher *PDP*.
- Optimizing the user experience when *Sources* with a higher *PDP Rating* are used with low power *Sinks*.
- Preventing *Sinks* that only function well (or at all) when using **Optional** voltages and currents.

10.3.2 Normative Sink Rules

Sinks designed to use *Sources* with a *PDP Rating* of x W **Shall**:

- Either operate or charge from *Sources* that have a *PDP Rating* $\geq x$ W.
- Either operate, charge or indicate a *Capabilities Mismatch* (see [Section 6.4.2.3, "Capability Mismatch"](#)) from *Sources* that have a *PDP Rating* $< x$ W and ≥ 0.5 W.

A *Sink* optimized for a *Source* with **Optional** voltages and currents or power as described in [Section 10.2.3, "Optional Voltages/Currents"](#) with a *PDP Rating* of x W **Shall** provide a similar user experience when powered from a *Source* with a *PDP Rating* of $\geq x$ W that supplies only the **Normative** voltages and currents as specified in [Section 10.2.2, "Normative Voltages and Currents"](#). For example, a 60W *Source* might not offer 9V Prog or 15V Prog since 20V Prog is a suitable substitute for both (as shown in [Table 10.10, "SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP"](#)).

The Operational Current/Power in the *Sink Capabilities* PDO for *Sinks* with an Operational *PDP* of x Watts **Shall** be as follows:

- Operational current for *Fixed Supply/Variable Supply* PDOs: RoundDown(x /voltage) to the nearest 10mA.
- Operational power for *Battery Supply* PDOs: $\leq x$.
- Operational current for Programmable Power Supply APDOs as defined in [Table 10.10, "SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP"](#): RoundDown (x /Prog Voltage) to the nearest 50mA.

The Maximum Current/Power in the *Sink RDO* for *Sinks* with an Operational *PDP* of x Watts and Maximum *PDP* of y Watts **Shall** be as follows:

- Maximum current for *Fixed Supply/Variable Supply* RDOs from *Sinks* without a *Battery*: RoundDown(x /voltage) to the nearest 10mA.
- Maximum current for *Fixed Supply/Variable Supply* RDOs from *Sinks* with a *Battery*: RoundDown(y /Voltage) to the nearest 10mA.
- Maximum power for *Battery Supply* RDOs from *Sinks* without a *Battery*: $\leq x$.
- Maximum power for *Battery Supply* RDOs from *Sinks* with a *Battery*: $\leq y$.
- Maximum current for PPS Supply RDOs from *Source* PDOs as defined in [Table 10.10, "SPR Programmable Power Supply PDOs and APDOs based on the Port Maximum PDP"](#) or [Table 10.14, "EPR Source Examples when Port Present PDP is less than Port Maximum PDP"](#): RoundDown (y /Prog Voltage) to the nearest 50mA.

The following requirements **Shall** apply to the *Advertised Sink Capabilities*:

- A Sink **Shall Not** Advertise Fixed Supply PDO maximum voltages and currents that exceed the *PDP Rating* they were designed to use.
- A Sink **Shall Not** Advertise Variable Supply PDO maximum voltages and currents that exceed the *PDP Rating* they were designed to use.
- A Sink **Shall Not** Advertise a Battery Supply PDO maximum allowable power that exceeds the *PDP Rating* they were designed to use.
- A Sink **Shall Not** Advertise a PPS APDO maximum allowable power that exceeds the *PDP Rating* they were designed to use.
- A Sink **Shall Not** Advertise an AVS APDO maximum allowable power that exceeds the *PDP Rating* they were designed to use.

A CRC calculation

A.1 C code example

```
//
// USB PD CRC Demo Code.
//
#include <stdio.h>
int crc;
//-----
void crcBits(int x, int len) {
    const int poly = 0x04C11DB6; //spec 04C1 1DB7h
    int newbit, newword, rl_crc;
    for(int i=0; i<len; i++) {
        newbit = ((crc>>31) ^ ((x>>i)&1)) & 1;
        if(newbit) newword=poly; else newword=0;
        rl_crc = (crc<<1) | newbit;
        crc = rl_crc ^ newword;
        printf("%2d newbit=%d, x>>i=0x%x, crc=0x%x\n", i, newbit, (x>>i), crc);
    }
}

int crcWrap(int c){
    int ret = 0;
    int j, bit;
    c = ~c;
    printf("~crc=0x%x\n", c);
    for(int i=0; i<32; i++) {
        j = 31-i;
        bit = (c>>i) & 1;
        ret |= bit<<j;
    }
    return ret;
}
//-----
int main(){
    int txCrc=0, rxCrc=0, residue=0, data;
    printf("using packet data 0x%x\n", data=0x0101);
    crc = 0xffffffff;
    crcBits(data, 16);
    txCrc = crcWrap(crc);
    printf("crc=0x%x, txCrc=0x%x\n", crc, txCrc);
    printf("received packet after decode= 0x%x, 0x%x\n", data, txCrc);
    crc = 0xffffffff;
    crcBits(data, 16);
    rxCrc = crcWrap(crc);
    printf("Crc of the received packet data is (of course) =0x%x\n", rxCrc);
    printf("continue by running the transmit crc through the crc\n");
    crcBits(rxCrc, 32);
    printf("Now the crc residue is 0x%x\n", crc);
    printf("should be 0xc704dd7b\n");
}
```

B Message Sequence Examples (Deprecated)

This appendix has been *Deprecated*.

C VDM Command Examples

C.1 Discover Identity Example

C.1.1 Discover Identity Command request

[Table C.1, "Discover Identity Command request from Initiator Example"](#) below shows the contents of the key fields in the *Message Header* and *VDM Header* for an *Initiator* sending a **Discover Identity** Command request.

Table C.1 Discover Identity Command request from Initiator Example

Bit(s)	Field	Value
Message Header		
15	<i>Reserved</i>	0
14...12	<i>Number of Data Objects</i>	1 (<i>VDM Header</i>)
11...9	<i>MessageID</i>	0...7
8	<i>Port Power Role</i>	0 or 1
7...6	<i>Specification Revision</i>	10b (<i>Revision 3.x</i>)
5...4	<i>Reserved</i>	0
3...0	<i>Message Type</i>	1111b (<i>Vendor Defined Message</i>)
VDM Header		
B31...16	<i>Standard or Vendor ID (SVID)</i>	0xFF00 (<i>PD SID</i>)
B15	<i>VDM Type</i>	1 (<i>Structured VDM</i>)
B14...13	<i>Structured VDM Version (Major)</i>	01b (<i>Version 2.0</i>)
B12...11	<i>Structured VDM Version (Minor)</i>	01b (<i>Version 2.1</i>)
B10...8	<i>Object Position</i>	000b
B7...6	<i>Command Type</i>	00b (<i>Initiator</i>)
B5	<i>Reserved</i>	0
B4...0	<i>Command</i>	1 (<i>Discover Identity</i>)

C.1.2 Discover Identity Command response - Active Cable.

Table C.2, "Discover Identity Command response from Active Cable Responder Example" shows the contents of the key fields in the *Message Header* and *VDM Header* for a *Responder* returning *VDOs* in response to a **Discover Identity** Command request. In this illustration, the *Responder* is an active Gen2 cable which supports *Modal Operation*.

Table C.2 Discover Identity Command response from Active Cable Responder Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	5 (VDM Header + ID Header VDO + Cert Stat VDO + Product VDO + Cable VDO)
11...9	MessageID	0...7
8	Cable Plug	1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	0xFF00 (PD SID)
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	000b
B7...6	Command Type	01b (Responder ACK)
B5	Reserved	0
B4...0	Command	2 (Discover Identity)
ID Header VDO		
B31	USB Communications Capable as USB Host	0 (not USB Communications capable as a USB Host)
B30	USB Communications Capable as a USB Device	0 (not data capable as a Device)
B29...27	SOP' Product Type (Cable Plug/VPD)	100b (Active Cable)
B26	Modal Operation Supported	1 (supports Modes)
B25...16	Reserved. Shall be set to zero.	0
B15...0	16-bit unsigned integer. USB Vendor ID	USB-IF assigned VID for this cable vendor
Cert Stat VDO		
B31...0	32-bit unsigned integer	USB-IF assigned XID for this cable
Product VDO		
B31...16	16-bit unsigned integer. USB Product ID	Product ID assigned by the cable vendor
B15...0	16-bit unsigned integer. bcdDevice	Device version assigned by the cable vendor
Cable VDO1 (returned for Product Type "Active Cable")		
B31...28	HW Version	Cable HW version number (vendor defined)
B27...24	Firmware Version	Cable FW version number (vendor defined)
B23...21	VDO Version	010b (Version 1.2)

Table C.2 Discover Identity Command response from Active Cable Responder Example

Bit(s)	Field	Value
B20	Reserved	0
B19...18	<i>USB Type-C plug to USB Type-C/Captive</i>	10b (<i>USB Type-C[®]</i>)
B17	<i>EPR Capable (Active Cable)</i>	0 (not <i>EPR Capable</i>)
B16...13	<i>Cable Latency</i>	0001b (<10ns (~1m))
B12...11	<i>Cable Termination Type (Active Cable)</i>	11b (Both ends Active, <i>VCONN</i> required)
B10...9	<i>Maximum VBUS Voltage (Active Cable)</i>	00b (20V)
B8	<i>SBU Supported</i>	0 (SBUs connections supported)
B7	<i>SBU Type</i>	0 (SBU is passive)
B6...5	<i>VBUS Current Handling Capability (Active Cable)</i>	01b (3A)
B4	<i>VBUS Through Cable</i>	1 (Yes)
B3	<i>SOP'' Controller Present</i>	1 (<i>SOP''</i> controller present)
B2...0	Reserved	0
Cable VDO2 (returned for Product Type "Active Cable")		
B31...24	<i>Maximum Operating Temperature</i>	70
B23...16	<i>Shutdown Temperature</i>	80
B15	Reserved	0
B14...12	<i>U3/CLd Power</i>	010b (1-5mW)
B11	<i>U3 to U0 transition mode</i>	00b (U3 to U0 direct)
B10	<i>Physical connection</i>	0 (Copper)
B9	<i>Active element</i>	0 (Active Re-driver)
B8	<i>USB4 Supported</i>	0 (USB4 Supported)
B7...6	<i>USB 2.0 Hub Hops Consumed</i>	2
B5	<i>USB 2.0 Supported</i>	0 (<i>[USB 2.0]</i> supported)
B4	<i>USB 3.2 Supported</i>	0 (<i>[USB 3.2]</i> SuperSpeed supported)
B3	<i>USB Lanes Supported</i>	1b (Two lanes)
B2	<i>Optically Isolated Active Cable</i>	0 (Not supported)
B1	<i>USB4 Asymmetric Mode Supported</i>	0 (Not Supported)
B0	<i>USB Gen</i>	1b (Gen 2 or higher)

C.1.3 Discover Identity Command response - Hub.

Table C.3, "Discover Identity Command response from Hub Responder Example" shows the contents of the key fields in the Message Header and VDM Header for a Responder returning VDOs in response to a **Discover SVIDs** Command request. In this illustration, the Responder is a Hub.

Table C.3 Discover Identity Command response from Hub Responder Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	4 (VDM Header + ID Header VDO + Cert Stat VDO + Product VDO)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	0xFF00 (PD SID)
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	000b
B7...6	Command Type	01b (Responder ACK)
B5	Reserved	0
B4...0	Command	2 (Discover Identity)
ID Header VDO		
B31	USB Communications Capable as USB Host	0 (not USB Communications capable as a USB Host)
B30	USB Communications Capable as a USB Device	1 (data capable as a Device)
B29...27	SOP' Product Type (Cable Plug/VPD)	001b (Hub)
B26	Modal Operation Supported	0 (doesn't support Modes)
B25...16	Reserved. Shall be set to zero.	0
B15...0	16-bit unsigned integer. USB Vendor ID	USB-IF assigned VID for this Hub vendor
Cert Stat VDO		
B31...0	32-bit unsigned integer	USB-IF assigned XID for this Hub
Product VDO		
B31...16	16-bit unsigned integer. USB Product ID	Product ID assigned by the Hub vendor
B15...0	16-bit unsigned integer. bcdDevice	Device version assigned by the Hub vendor

C.2 Discover SVIDs Example

C.2.1 Discover SVIDs Command request

[Table C.4, "Discover SVIDs Command request from Initiator Example"](#) below shows the contents of the key fields in the *Message Header* and *VDM Header* for an *Initiator* sending a **Discover SVIDs** Command request.

Table C.4 Discover SVIDs Command request from Initiator Example

Bit(s)	Field	Value
Message Header		
15	<i>Reserved</i>	0
14...12	<i>Number of Data Objects</i>	1 (<i>VDM Header</i>)
11...9	<i>MessageID</i>	0...7
8	<i>Port Power Role</i>	0 or 1
7...6	<i>Specification Revision</i>	10b (<i>Revision 3.x</i>)
5...4	<i>Reserved</i>	0
3...0	<i>Message Type</i>	1111b (<i>Vendor Defined Message</i>)
VDM Header		
B31...16	<i>Standard or Vendor ID (SVID)</i>	0xFF00 (<i>PD SID</i>)
B15	<i>VDM Type</i>	1 (<i>Structured VDM</i>)
B14...13	<i>Structured VDM Version (Major)</i>	01b (<i>Version 2.0</i>)
B12...11	<i>Structured VDM Version (Minor)</i>	01b (<i>Version 2.1</i>)
B10...8	<i>Object Position</i>	000b
B7...6	<i>Command Type</i>	00b (<i>Initiator</i>)
B5	<i>Reserved</i>	0
B4...0	<i>Command</i>	2 (<i>Discover SVIDs</i>)

C.2.2 Discover SVIDs Command response

[Table C.5, "Discover SVIDs Command response from Responder Example"](#) shows the contents of the key fields in the *Message Header* and *VDM Header* for a *Responder* returning VDOs in response to a **Discover SVIDs Command** request. In this illustration, the value 3 in the Message Header **Number of Data Objects** field indicates that one VDO containing the supported SVIDs would be returned followed by a terminating VDO.

Note: The last VDO returned (the terminator of the transmission) contains zero value SVIDs. If a SVID value is zero, it is not used.

Table C.5 Discover SVIDs Command response from Responder Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	3 (VDM Header + 2*VDO)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	0xFF00 (PD SID)
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	000b
B7...6	Command Type	01b (Responder ACK)
B5	Reserved	0
B4...0	Command	2 (Discover SVIDs)
VDO 1		
B31...16	SVID 0	SVID value
B15...0	SVID 1	SVID value
VDO 2		
B31...16	SVID 2	0x0000
B15...0	SVID 3	0x0000

C.3 Discover Modes Example

C.3.1 Discover Modes Command request

[Table C.6, "Discover Modes Command request from Initiator Example"](#) shows the contents of the key fields in the Message Header and VDM Header for an *Initiator* sending a **Discover Modes** Command request. The *Initiator* of the **Discover Modes** Command AMS sends a Message Header with the **Number of Data Objects** field set to 1 followed by a VDM Header with the **Command Type** (B7...6) set to zero indicating the *Command* is from an *Initiator* and the **Command** (B4...0) is set to 3 indicating Mode discovery.

Table C.6 Discover Modes Command request from Initiator Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	1 (VDM Header)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for which Modes are being requested
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	000b
B7...6	Command Type	00b (Initiator)
B5	Reserved	0
B4...0	Command	3 (Discover Modes)

C.3.2 Discover Modes Command response

The *Responder* to the **Discover Modes Command** request returns a Message Header with the **Number of Data Objects** field set to a value of 1 to 7 (the actual value is the number of Mode objects plus one) followed by a VDM Header with the **Command Type** (B7...6) set to 1 indicating the *Command* is from a *Responder* and the **Command** (B4...0) set to 3 indicating the following objects describe the Modes the device supports. If the ID is a VID, the structure and content of the VDO is left to the vendor. If the ID is a SID, the structure and content of the VDO is defined by the Standard.

[Table C.7, "Discover Modes Command response from Responder Example"](#) shows the contents of the key fields in the Message Header and VDM Header for a *Responder* returning VDOs in response to a **Discover Modes Command** request. In this illustration, the value 2 in the Message Header **Number of Data Objects** field indicates that the device is returning one VDO describing the Mode it supports. It is possible for a *Responder* to report up to six different Modes.

Table C.7 Discover Modes Command response from Responder Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	2 (VDM Header + 1 Mode VDO)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for which Modes were requested
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	000b
B7...6	Command Type	01b (Responder ACK)
B5	Reserved	0
B4...0	Command	3 (Discover Modes)
Mode VDO		
B31...0	Mode 1	Standard or Vendor defined Mode value

C.4 Enter Mode Example

C.4.1 Enter Mode Command request

The *Initiator* of the **Enter Mode Command** request sends a Message Header with the **Number of Data Objects** field set to 1 followed by a VDM Header with the **Command Type** (B7...6) set to 0 indicating the *Command* is from an *Initiator* and the **Command** (B4...0) set to 4 to request the *Responder* to enter its mode of operation and sets the Object Position field to the desired functional VDO based on its offset as received during Discovery.

[Table C.8, "Enter Mode Command request from Initiator Example"](#) shows the contents of the key fields in the Message Header and VDM Header for an *Initiator* sending an **Enter Mode Command** request.

Table C.8 Enter Mode Command request from Initiator Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	1 (VDM Header)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for the Mode being entered
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	001b (a one in this field indicates a request to enter the first Mode in list returned by Discover Modes)
B7...6	Command Type	00b (Initiator)
B5	Reserved	0
B4...0	Command	4 (Enter Mode)

C.4.2 Enter Mode Command response

The *Responder* that is the target of the *Enter Mode Command* request sends a Message Header with the *Number of Data Objects* field set to a value of 1 followed by a VDM Header with the *Command Type* (B7...6) set to 1 indicating the *Command* is from a *Responder* and the *Command* (B4...0) set to 4 indicating the *Responder* has entered the Mode and is ready to operate.

[Table C.9, "Enter Mode Command response from Responder Example"](#) shows the contents of the key fields in the Message Header and VDM Header for a *Responder* sending an *Enter Mode Command* response with an *ACK*.

Table C.9 Enter Mode Command response from Responder Example

Bit(s)	Field	Value
Message Header		
15	<i>Reserved</i>	0
14...12	<i>Number of Data Objects</i>	1 (VDM Header)
11...9	<i>MessageID</i>	0...7
8	<i>Port Power Role</i>	0 or 1
7...6	<i>Specification Revision</i>	10b (Revision 3.x)
5...4	<i>Reserved</i>	0
3...0	<i>Message Type</i>	1111b (Vendor Defined Message)
VDM Header		
B31...16	<i>Standard or Vendor ID (SVID)</i>	SVID for the Mode entered
B15	<i>VDM Type</i>	1 (Structured VDM)
B14...13	<i>Structured VDM Version (Major)</i>	01b (Version 2.0)
B12...11	<i>Structured VDM Version (Minor)</i>	01b (Version 2.1)
B10...8	<i>Object Position</i>	001b (offset of the Mode entered)
B7...6	<i>Command Type</i>	01b (Responder <i>ACK</i>)
B5	<i>Reserved</i>	0
B4...0	<i>Command</i>	4 (<i>Enter Mode</i>)

C.4.3 Enter Mode Command request with additional VDO.

The *Initiator* of the **Enter Mode Command** request sends a Message Header with the **Number of Data Objects** field set to 2 indicating an additional VDO followed by a VDM Header with the **Command Type** (B7...6) set to zero indicating the *Command* is from an *Initiator* and the **Command** (B4...0) set to 4 to request the *Responder* to enter its mode of operation and sets the Object Position field to the desired functional VDO based on its offset as received during Discovery.

[Table C.10, "Enter Mode Command request with additional VDO Example"](#) shows the contents of the key fields in the Message Header and VDM Header for an *Initiator* sending an **Enter Mode Command** request with an additional VDO.

Table C.10 Enter Mode Command request with additional VDO Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	1 (VDM Header)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for the Mode being entered
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	001b (a one in this field indicates a request to enter the first Mode in list returned by Discover Modes)
B7...6	Command Type	00b (Initiator)
B5	Reserved	0
B4...0	Command	4 (Enter Mode)
Including Optional Mode specific VDO		
B31...0	Mode specific	

C.5 Exit Mode Example

C.5.1 Exit Mode Command request

The *Initiator* of the **Exit Mode Command** request sends a Message Header with the **Number of Data Objects** field set to 1 followed by a VDM Header with the **Command Type** (B7...6) set to zero indicating the *Command* is from an *Initiator* and the **Command** (B4...0) set to 5 to request the *Responder* to exit its Mode of operation.

[Table C.11, "Exit Mode Command request from Initiator Example"](#) shows the contents of the key fields in the *Message Header* and *VDM Header* for an *Initiator* sending an **Exit Mode Command** request.

Table C.11 Exit Mode Command request from Initiator Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	1 (VDM Header)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for the Mode being exited
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	001b (identifies the previously entered Mode by its Object Position that is to be exited)
B7...6	Command Type	00b (Initiator)
B5	Reserved	0
B4...0	Command	5 (Exit Mode)

C.5.2 Exit Mode Command response

The *Responder* that receives the *Exit Mode Command* request sends a Message Header with the *Number of Data Objects* field set to a value of 1 followed by a VDM Header with the *Command Type* (B7...6) set to 1 indicating the *Command* is from a *Responder* and the *Command* (B4...0) set to 5 indicating the *Responder* has exited the Mode and has returned to normal USB operation.

[Table C.12, "Exit Mode Command response from Responder Example"](#) shows the contents of the key fields in the Message Header and VDM Header for a *Responder* sending an *Exit Mode Command ACK* response.

Table C.12 Exit Mode Command response from Responder Example

Bit(s)	Field	Value
Message Header		
15	<i>Reserved</i>	0
14...12	<i>Number of Data Objects</i>	1 (VDM Header)
11...9	<i>MessageID</i>	0...7
8	<i>Port Power Role</i>	0 or 1
7...6	<i>Specification Revision</i>	10b (Revision 3.x)
5...4	<i>Reserved</i>	0
3...0	<i>Message Type</i>	1111b (Vendor Defined Message)
VDM Header		
B31...16	<i>Standard or Vendor ID (SVID)</i>	SVID for the Mode exited
B15	<i>VDM Type</i>	1 (Structured VDM)
B14...13	<i>Structured VDM Version (Major)</i>	01b (Version 2.0)
B12...11	<i>Structured VDM Version (Minor)</i>	01b (Version 2.1)
B10...8	<i>Object Position</i>	001b (offset of the Mode to be exited)
B7...6	<i>Command Type</i>	01b (<i>Responder ACK</i>)
B5	<i>Reserved</i>	0
B4...0	<i>Command</i>	5 (<i>Exit Mode</i>)

C.6 Attention Example

C.6.1 Attention Command request

The *Initiator* of the **Attention Command** request sends a Message Header with the **Number of Data Objects** field set to 1 followed by a VDM Header with the **Command Type** (B7...6) set to zero indicating the *Command* is from an *Initiator* and the **Command** (B4...0) set to 6 to request attention from the *Responder*.

[Table C.13, "Attention Command request from Initiator Example"](#) shows the contents of the key fields in the *Message Header* and *VDM Header* for an *Initiator* sending an **Attention Command** request.

Table C.13 Attention Command request from Initiator Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	1 (<i>VDM Header</i>)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (<i>Revision 3.x</i>)
5...4	Reserved	0
3...0	Message Type	1111b (<i>Vendor Defined Message</i>)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for which attention is being requested
B15	VDM Type	1 (<i>Structured VDM</i>)
B14...13	Structured VDM Version (Major)	01b (<i>Version 2.0</i>)
B12...11	Structured VDM Version (Minor)	01b (<i>Version 2.1</i>)
B10...8	Object Position	001b (offset of the Mode requesting attention)
B7...6	Command Type	00b (<i>Initiator</i>)
B5	Reserved	0
B4...0	Command	6 (Attention)

C.6.2 Attention Command request with additional VDO.

The *Initiator* of the **Attention Command** request sends a Message Header with the **Number of Data Objects** field set to 2 indicating an additional VDO followed by a VDM Header with the **Command Type** (B7...6) set to zero indicating the *Command* is from a *Responder* and the **Command** (B4...0) set to 6 to request attention from the *Responder*.

[Table C.14, "Attention Command request from Initiator with additional VDO Example"](#) shows the contents of the key fields in the *Message Header* and *VDM Header* for an *Initiator* sending an **Attention Command** request with an additional VDO.

Table C.14 Attention Command request from Initiator with additional VDO Example

Bit(s)	Field	Value
Message Header		
15	Reserved	0
14...12	Number of Data Objects	2 (VDM Header + VDO)
11...9	MessageID	0...7
8	Port Power Role	0 or 1
7...6	Specification Revision	10b (Revision 3.x)
5...4	Reserved	0
3...0	Message Type	1111b (Vendor Defined Message)
VDM Header		
B31...16	Standard or Vendor ID (SVID)	SVID for which attention is being requested
B15	VDM Type	1 (Structured VDM)
B14...13	Structured VDM Version (Major)	01b (Version 2.0)
B12...11	Structured VDM Version (Minor)	01b (Version 2.1)
B10...8	Object Position	001b (offset of the Mode requesting attention)
B7...6	Command Type	000b (Initiator)
B5	Reserved	0
B4...0	Command	6 (Attention)
Including Optional Mode specific VDO		
B31...0	Mode specific	

D BMC Receiver Design Examples

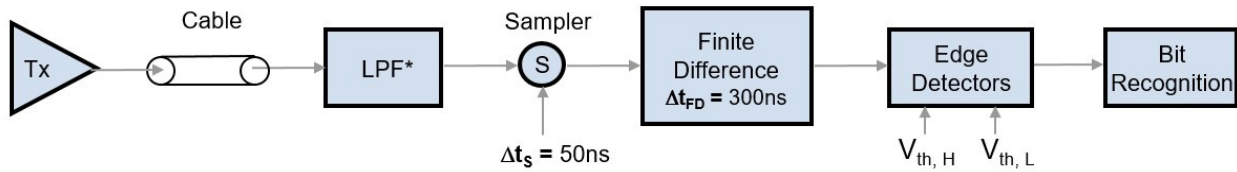
The *BMC* signal is DC-coupled so that the voltage level is affected by the ground *IR Drop*. The DC offset of the *BMC* signal at Power Source and Power *Sink* are in the opposite directions. When the V_{BUS} current is increased from 0A, the *BMC* signal waveform shifts downward at Power *Sink* and shifts upward at Power Source. This section introduces two sample *BMC* receiver circuit implementations, which are immune from DC offset and high current load step. They can be used in Power Source, Power *Sink* and inside cables.

D.1 Finite Difference Scheme

D.1.1 Sample Circuitry

The sample Finite Difference *BMC* receiver shown in [Figure D.1, "Circuit Block of BMC Finite Difference Receiver"](#) consists of the Rx bandwidth limiting filter with the time constant $t_{RxFilter}$, a sampler with the sampling step Δt_s , 50ns, a Finite Difference Calculator which calculates the voltage difference between the time interval of Δt_{FD} , 300ns, an edge detector controlled by two voltage thresholds, $V_{th,H}$ and $V_{th,L}$ and a logic block for bit recognition.

Figure D.1 Circuit Block of BMC Finite Difference Receiver



D.1.2 Theory

This section describes the fundamental theory of Finite Difference Scheme to recover the received *BMC* signal with the input and output signal waveforms of the circuit blocks shown in [Figure D.1, "Circuit Block of BMC Finite Difference Receiver"](#). To illustrate the robustness of the implementation, the V_{BUS} current load step rate is intentionally increased to 2A/ μ s at the *Sink* load. In [Figure D.2, "BMC AC and DC noise from \$V_{BUS}\$ at Power Sink"](#) (a), the red curve represents the V_{BUS} current measured at the Power *Sink* when the current is increased at 9 μ s from 0A to 5A and the blue dash curve represents the V_{BUS} current measured at the USB Type-C[®] connector of the power *Sink*. In this example, the peak current overshoot with larger load step rate is increased to 518 mA which exceeds *iOvershoot*. [Figure D.2, "BMC AC and DC noise from \$V_{BUS}\$ at Power Sink"](#) (b) shows the total *BMC* noise at Power *Sink*, coupled from V_{BUS} and D+/D- through the worst [\[USB Type-C 2.4\]](#) compliant cable, after the Rx bandwidth limiting filter with the time constant $t_{RxFilter}$ is applied. The noise can be decomposed into 3 components. The first is the DC offset, $I_{VBUS}(t) \cdot R_{GND}$, while I_{VBUS} is the V_{BUS} current and R_{GND} is the ground DC resistance of the cable. The offset is negative in Power *Sink* and positive at Power Source. The second noise component is the inductive V_{BUS} noise, $M \cdot dI_{VBUS}(t)/dt$, while M is the mutual inductance between the V_{BUS} and CC wires in the cable and $dI_{VBUS}(t)/dt$ is the load step rate. The third component is [\[USB 2.0\]](#) Full Speed SE0 coupling noise which would normally occur randomly but was assumed to occur periodically in the simulation to account for the crosstalk in any phase between the *BMC* and [\[USB 2.0\]](#) signals. In [Figure D.3, "Sample BMC Signals \(a\) without USB 2.0 SE0 Noise \(b\) with USB 2.0 SE0 Noise"](#), the blue dash curve represents the *BMC* signal when there is no V_{BUS} current, and the red solid curve represents the *BMC* signal affected by the V_{BUS} coupling noise shown in [Figure D.2, "BMC AC and DC noise from \$V_{BUS}\$ at Power Sink"](#) (b). The green solid curve is the sample [\[USB 2.0\]](#) noise, after the Rx bandwidth limiting filter with the time constant $t_{RxFilter}$ is applied.

Figure D.2 BMC AC and DC noise from VBUS at Power Sink

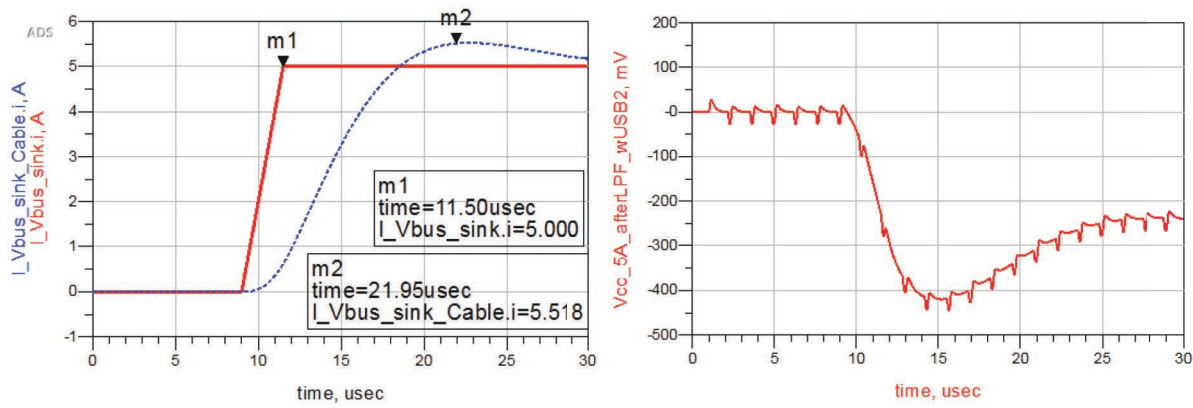
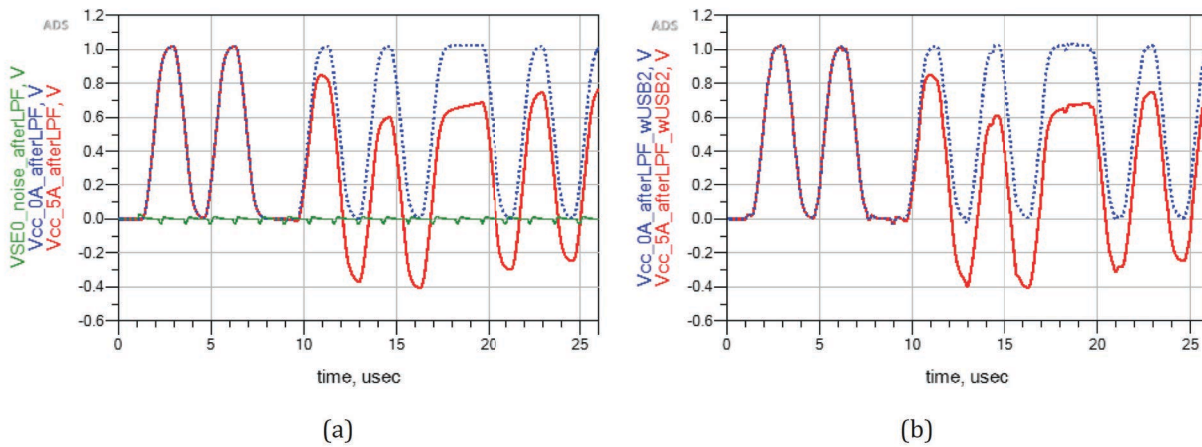
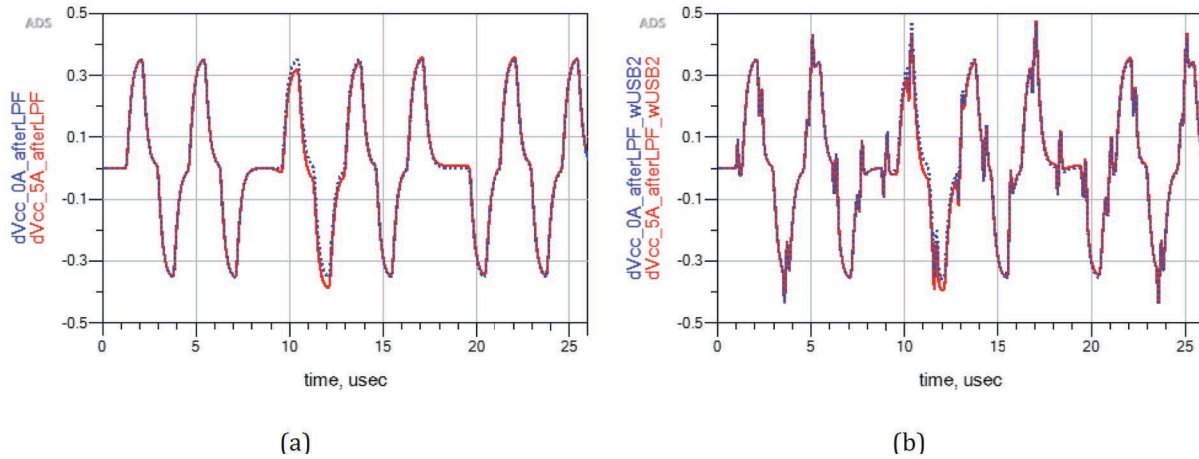


Figure D.3 Sample BMC Signals (a) without USB 2.0 SE0 Noise (b) with USB 2.0 SE0 Noise



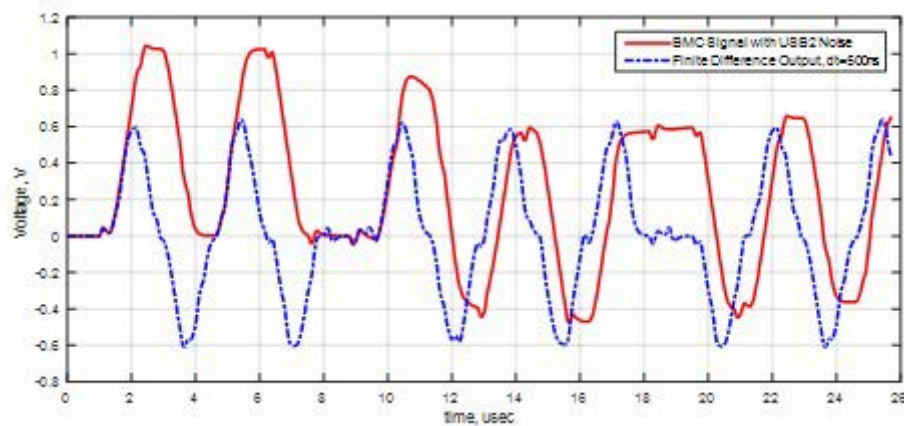
The BMC signals shown in [Figure D.3, "Sample BMC Signals \(a\) without USB 2.0 SE0 Noise \(b\) with USB 2.0 SE0 Noise"](#) are sampled every 50ns and the scaled derivative waveforms, $V_{cc}(t) - V_{cc}(t - 50ns)$, without and with [\[USB 2.0\]](#) noise are shown in [Figure D.4, "Scaled BMC Signal Derivative with 50ns Sampling Rate \(a\) without USB 2.0 Noise \(b\) with USB 2.0 Noise"](#) (a) and (b), respectively. In [Figure D.4, "Scaled BMC Signal Derivative with 50ns Sampling Rate \(a\) without USB 2.0 Noise \(b\) with USB 2.0 Noise"](#) (a), if there is no [\[USB 2.0\]](#) noise, the derivative waveform just changes slightly before and after the V_{BUS} current transition. That means, the slope of the BMC waveform is not sensitive to the DC offset and is very useful to be used to design a robust receiver against a large DC offset. However, the derivative waveforms with [\[USB 2.0\]](#) noise have large perturbation as shown in [Figure D.4, "Scaled BMC Signal Derivative with 50ns Sampling Rate \(a\) without USB 2.0 Noise \(b\) with USB 2.0 Noise"](#) (b).

Figure D.4 Scaled BMC Signal Derivative with 50ns Sampling Rate (a) without USB 2.0 Noise (b) with USB 2.0 Noise



To remove the high frequency content of the [\[USB 2.0\]](#) noise, Finite Difference technique with the proper time interval is applied to the *BMC* waveform with [\[USB 2.0\]](#) noise in [Figure D.3, "Sample BMC Signals \(a\) without USB 2.0 SE0 Noise \(b\) with USB 2.0 SE0 Noise"](#). Using Backward Finite Difference Calculator, $\Delta V_{cc} = V_{cc}(t) - V_{cc}(t - \Delta t)$, [Figure D.5, "BMC Signal and Finite Difference Output with Various Time Steps"](#) shows the Finite Difference Output while $\Delta t = 500\text{ns}$. The larger the time interval Δt is, the larger the peak-to-peak magnitude of the Finite Difference Output will be. However, the time interval is bounded by the rise time of the *BMC* signal so that 300ns to 500ns is a good range of the time interval.

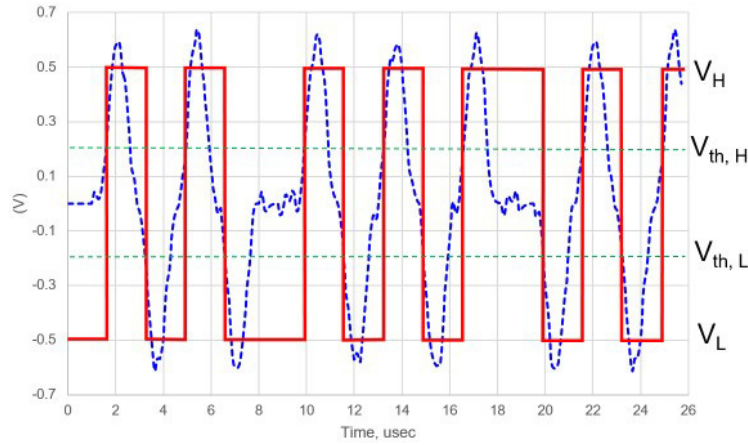
Figure D.5 BMC Signal and Finite Difference Output with Various Time Steps



D.1.3 Data Recovery

The edge detection is followed by the Finite Difference Calculation. At the input of the edge detector, if the voltage is larger than $V_{th,H}$ at the rising edge, the output will become high voltage level, V_H , if the voltage is smaller than $V_{th,L}$ at the falling edge, the output will become low voltage level, V_L . In this example, $V_{th,H}$ and $V_{th,L}$ are 0.2V and -0.2V, respectively. The solid curve in [Figure D.6, "Output of Finite Difference in dash line and Edge Detector in solid line"](#) represents the output of the edge detector, where V_H is 0.5V and V_L is -0.5V.

Figure D.6 Output of Finite Difference in dash line and Edge Detector in solid line



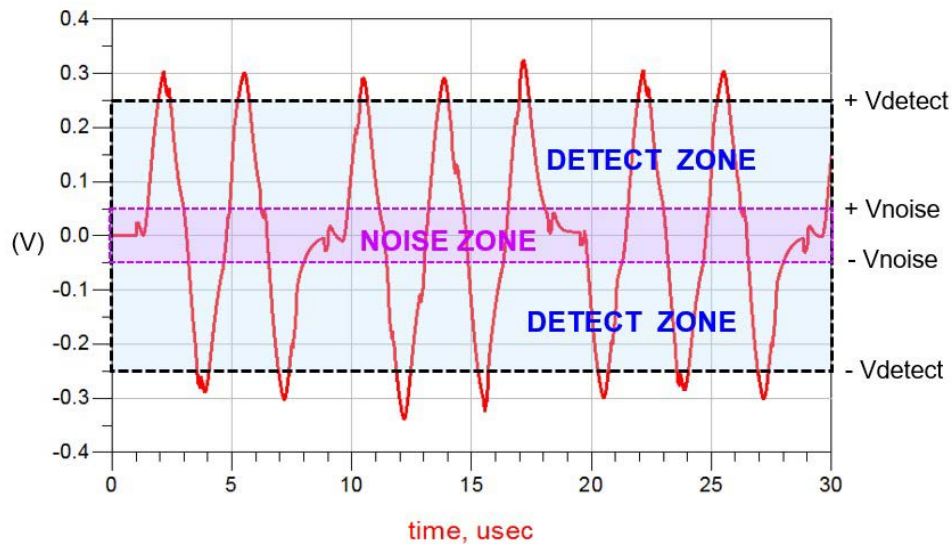
The duty cycle of the output signal from the edge detector varies depending on the thresholds, $V_{th,H}$ and $V_{th,L}$, as well as jitter and noise from silicon and channel. The techniques such as integrating receiver can be used to recover the *BMC* signal.

D.1.4 Noise Zone and Detection Zone

[Figure D.7, "Noise Zone and Detect Zone of BMC Receiver"](#) shows the output of Finite Difference when the time interval of Finite Difference is set to 300ns. The noise Zone is defined in between $+V_{noise}$ and $-V_{noise}$, in which the noise glitches occur. The detect zone is defined in between $+V_{detect}$ and $-V_{detect}$, excluding the noise zone. The thresholds of the edge detectors, $V_{th,H}$ and $V_{th,L}$, must be properly set within the detect zone so that the data can be recovered successfully.

In this example, V_{detect} is 250mV and V_{noise} is 50mV. It is highly recommended that the product implemented with the similar techniques indicates the performance with the range of V_{noise} and V_{detect} in the electrical specification.

Figure D.7 Noise Zone and Detect Zone of BMC Receiver

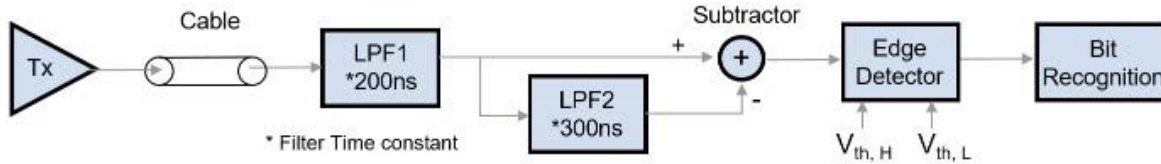


D.2 Subtraction Scheme

D.2.1 Sample Circuitry

The sample Subtraction BMC receiver shown in [Figure D.8, "Circuit Block of BMC Subtraction Receiver"](#) consists of the two Low Pass Filters (LPF1 and LPF2), a Subtractor, an Edge Detector and a logic block for bit recognition. The time constant of the first and second LPF are 200ns and 300ns, respectively. The Subtractor subtracts the LPF1 output from the LPF2 output. The Edge Detector controlled by two voltage thresholds, $V_{th,H}$ and $V_{th,L}$ to recover the data.

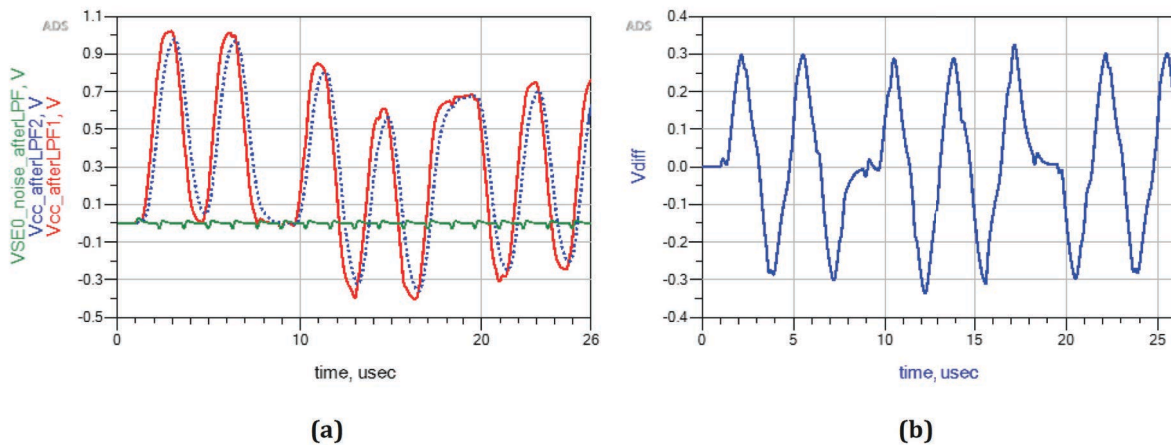
Figure D.8 Circuit Block of BMC Subtraction Receiver



D.2.2 Output of Each Circuit Block

[Figure D.9, "\(a\) Output of LPF1 and LPF2 \(b\) Subtraction of LPF1 and LPF2 Output"](#) (a) shows the output of LPF1 as the red solid line and LPF2 as the blue dash line as well as the [\[USB 2.0\]](#) noise in green solid line. [Figure D.9, "\(a\) Output of LPF1 and LPF2 \(b\) Subtraction of LPF1 and LPF2 Output"](#) (b) shows the voltage difference between the two output filters, $V_{diff} = V_{cc_afterLPF1} - V_{cc_afterLPF2}$. The V_{diff} waveform looks very similar to the Finite Difference output waveform shown in [Figure D.6, "Output of Finite Difference in dash line and Edge Detector in solid line"](#) so that the data recovery method through the edge detector is the same as described in [Section D.1.3, "Data Recovery"](#).

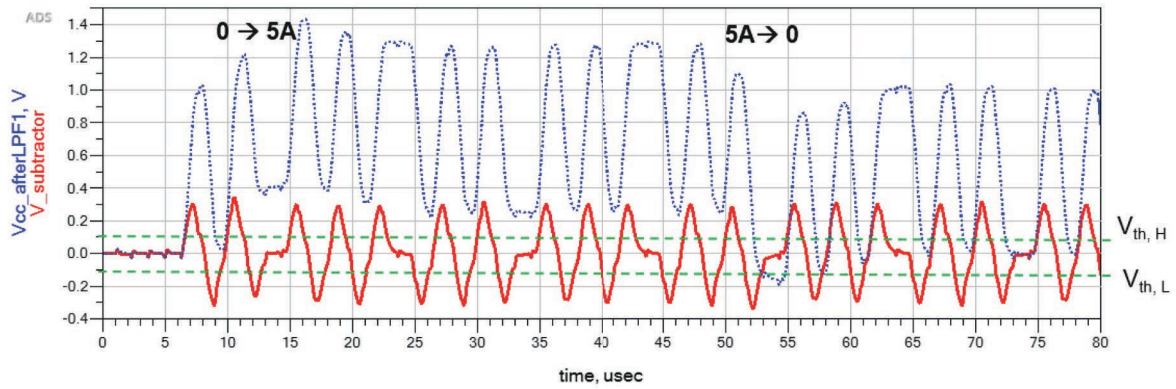
Figure D.9 (a) Output of LPF1 and LPF2 (b) Subtraction of LPF1 and LPF2 Output



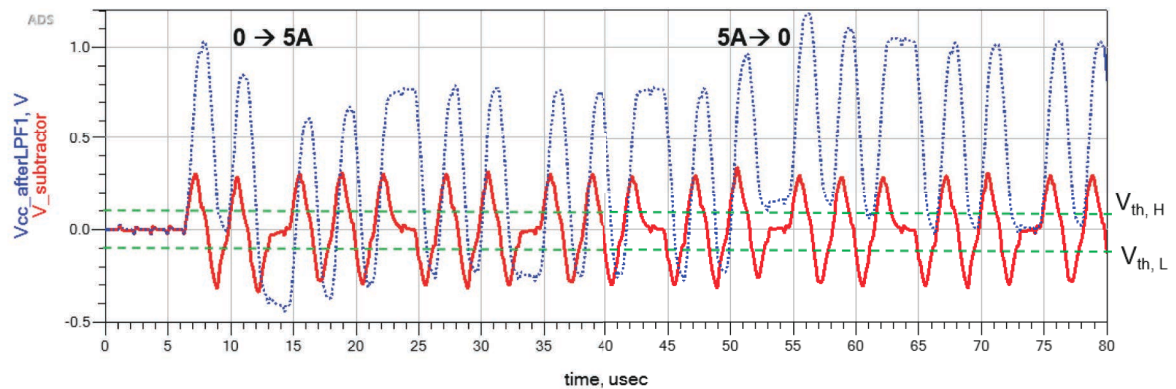
D.2.3 Subtractor Output at Power Source and Power Sink

The following figures show the example when the V_{BUS} current increases from 0A to 5A and then decreases to 0A with high load step rate. The output of the LPF1 and the Subtractor at Power Source and Power Sink are shown in [Figure D.10, "Output of the BMC LPF1 in blue dash curve and the Subtractor in red solid curve \(a\) at Power Source \(b\) at Power Sink"](#) (a) and (b), respectively. Although the BMC signals at Power Source and Power Sink shift toward the opposite direction, the Subtractor outputs at Power Source and Power Sink are almost identical regardless of the opposite direction of the DC offset.

Figure D.10 Output of the BMC LPF1 in blue dash curve and the Subtractor in red solid curve (a) at Power Source (b) at Power Sink



(a)



(b)

D.2.4 Noise Zone and Detection Zone

The zone definition is the same as defined in [Section D.1.4, "Noise Zone and Detection Zone"](#). The sizes of the noise zone and detection zone of the Subtraction Scheme are dependent on the filter time constant. When the time constant of the first and second LPF are 200ns and 300ns, respectively, V_{detect} is 250mV and V_{noise} is 50mV. It is highly recommended that the product implemented with the similar techniques indicates the performance with the range of V_{noise} and V_{detect} in the electrical specification.

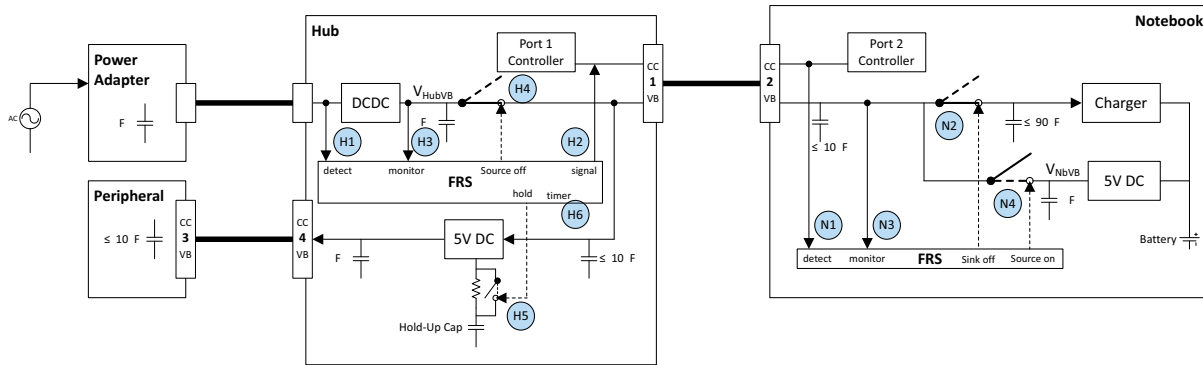
E FRS System Level Example

E.1 Overview

[Appendix E, "FRS System Level Example"](#) is intended to clarify *Fast Role Swap* (FRS) functionality at the system level through the use of an example implementation.

[Figure E.1, "Example FRS Capable System"](#) is an example of a *Hub* and laptop implementation that supports *Fast Role Swap* (see [Figure 7.16, "VBUS Power during Fast Role Swap"](#)). It is not the only possible *Hub* or laptop architecture. However, it is intended to provide an example system whose functionality is used here to illustrate how *Fast Role Swap* works.

Figure E.1 Example FRS Capable System



This appendix describes two cases that cover a variety of behaviors that might be seen in practice.

- **Slow *VBUS* Discharge** where *VBUS* between the *Hub* and the laptop takes more than 15ms ($t_{FRSwapInit}$) to discharge below 5.5 V (v_{Safe5V} (max)). In this case the **FR_Swap** Message is sent by the laptop while *VBUS* is still greater than v_{Safe5V} (max). See [Figure E.2, "Slow *VBUS* Discharge"](#).
- **Fast *VBUS* Discharge** where *VBUS* between the *Hub* and the laptop discharges very quickly, perhaps before the *Fast Role Swap Request* is even complete. See [Figure E.3, "Fast *VBUS* Discharge"](#).

However, neither the *Hub* nor the laptop can anticipate how quickly *VBUS* will discharge until the power adapter is disconnected from an *AC Supply* or it is unplugged from the *Hub*.

The *Fast Role Swap Request* is the momentary low driven by the *Hub* on the *CC* wire which is detected by the laptop.

[Figure E.2, "Slow *VBUS* Discharge"](#) and [Figure E.3, "Fast *VBUS* Discharge"](#) show the voltage seen on *VBUS* in relationship to the *Fast Role Swap Request*. They also show the transition between when the *Hub* stops supplying *VBUS* and when the laptop starts supplying *VBUS*.

Figure E.2 Slow VBUS Discharge

VBUS voltage when it discharges slowly
(assume very small cable IR drop prior to FR swap)

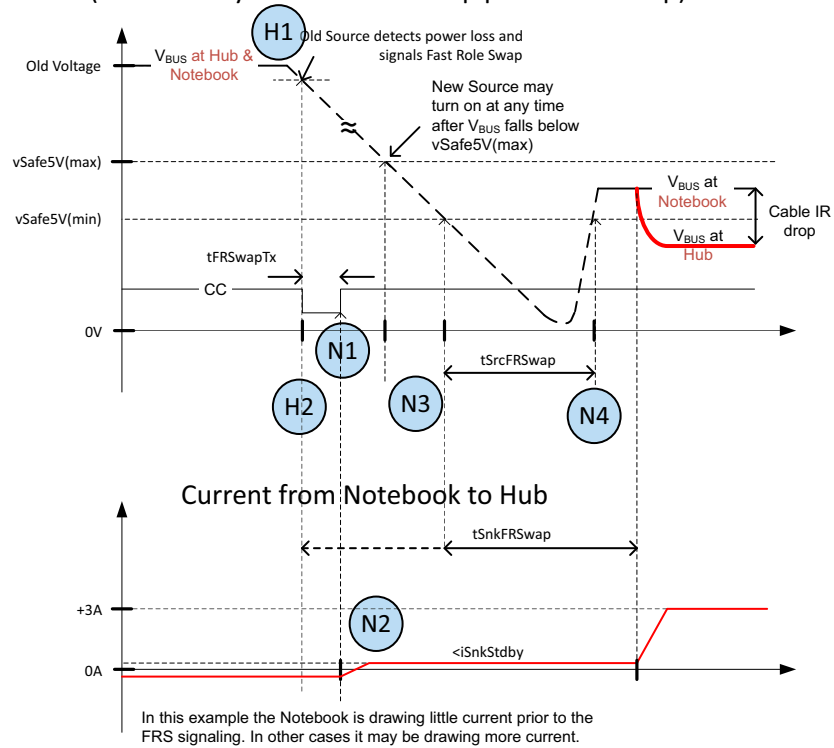
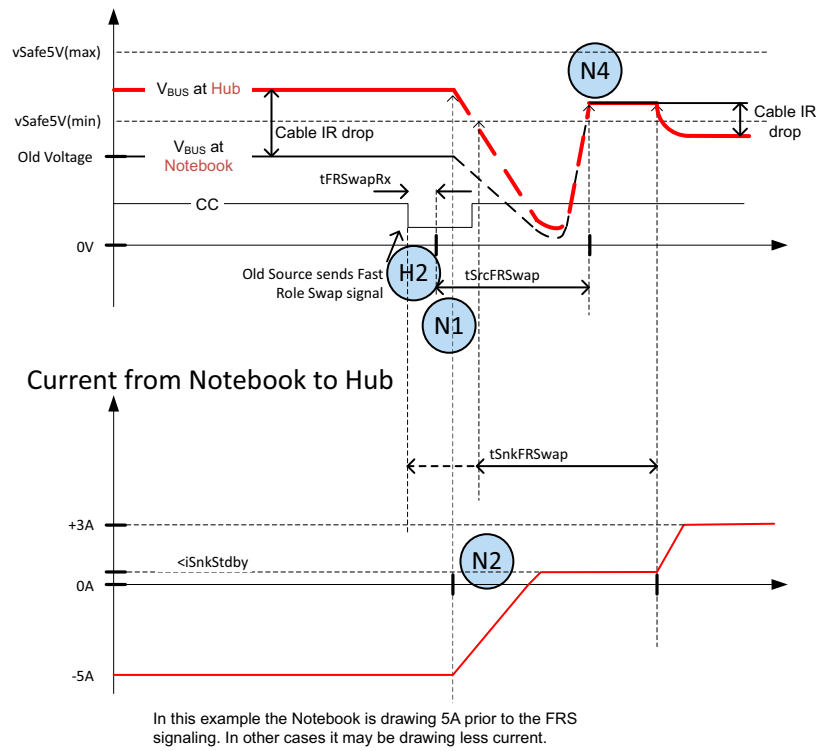


Figure E.3 Fast VBUS Discharge

VBUS voltage when it discharges quickly



E.2 FRS Initial Setup

Before a *Fast Role Swap* can occur, some initial setup steps are required. They require the laptop to discover whether *Fast Role Swap* is supported by the *Hub*, the amount of current the *Hub* requires after a *Fast Role Swap*, and whether the laptop is able and willing to provide that amount. They also ensure that the laptop supplies *VCONN* before, during and after an FRS. [Table E.1, "Sequence for setup of a Fast Role Swap \(Hub connected to Power Adapter first\)"](#) and [Table E.2, "Sequence for setup of a Fast Role Swap \(Hub connected to laptop before Power Adapter\)"](#) below show two typical sequences that might be used to prepare a laptop to support *Fast Role Swap*.

Table E.1 Sequence for setup of a Fast Role Swap (Hub connected to Power Adapter first)

Step #	Hub	Laptop
1	Hub connected to power adapter	
2	Hub is connected to laptop.	
3		Laptop sources 5 V to <i>VBUS</i> (vSafe5V). Laptop sources 5 V to <i>VCONN</i>
4		Laptop reads the cable to check its current carrying capability and/or if it is an <i>Active Cable</i> requiring <i>VCONN</i> .
5		Laptop sends a <i>Capabilities Message</i>
6	Hub sends a Request Message	
7	Hub and laptop establish an <i>Explicit Contract</i> with Hub as Sink.	
8		Laptop sends a Get_Source_Cap Message to determine how much power the Hub can provide.
9	Hub sends a Source_Capabilities Message with the Dual-Role Power bit set, and Unconstrained Power bit set, and Maximum Current > 0.	
10		Since the Hub can supply power the laptop sends a PR_Swap Message
11	Hub sends an Accept Message and starts supplying <i>VBUS</i>	
12		Laptop sends a Get_Sink_Cap Message to determine the current required by the Hub to support an FRS. If the Hub does not support FRS or the laptop cannot supply the required current, the laptop Ignores any <i>Fast Role Swap Requests</i> it might see.
13	If the Hub can supply more than 3A, it initiates a <i>VCONN Swap</i> to make to make itself the <i>VCONN Source</i> and reads the cable to check its current carrying capability.	
14	Hub sends a Sink_Capabilities Message	
15		Laptop sends a Request Message
16	Hub and laptop establish an <i>Explicit Contract</i> with Hub as source.	

Table E.1 Sequence for setup of a Fast Role Swap (Hub connected to Power Adapter first)

Step #	Hub	Laptop
17		If the laptop has detected that it is connected via an <i>Active Cable</i> (or one that supports <i>Alternate Modes</i>) and/or that it can support an <i>FRS</i> , it initiates a <i>VCONN Swap</i> to make itself the <i>VCONN Source</i> . This removes a requirement that the <i>Hub</i> to hold up <i>VCONN</i> during the <i>FRS</i> .
18	Normal PD Power traffic flow	
19	The <i>Hub</i> and laptop are now ready to do a <i>Fast Role Swap</i> in case the power adapter gets removed.	

Table E.2 Sequence for setup of a Fast Role Swap (Hub connected to laptop before Power Adapter)

Step #	Hub	Laptop
0	<i>Hub</i> is connected to laptop.	
1		Laptop sources 5 V to <i>VBUS</i> (vSafe5V). Laptop sources 5 V to <i>VCONN</i>
2		Laptop reads the cable to check its current carrying capability and/or if it is an <i>Active Cable</i> requiring <i>VCONN</i> .
3		Laptop sends Source_Capabilities Message
4	<i>Hub</i> sends Request Message	
5	<i>Hub</i> and laptop establish an <i>Explicit Contract</i> with <i>Hub</i> as <i>Sink</i> .	
6		Laptop sends a Get_Source_Cap Message to determine how much power the <i>Hub</i> can provide
7	<i>Hub</i> sends a Source_Capabilities Message with the Dual-Role Power bit set, and Unconstrained Power bit cleared, and Maximum Current = 0.	
8		Since the <i>Hub</i> cannot supply power, the laptop does not send a PR_Swap Message
9	The power adapter is connected to the <i>Hub</i>	
10	If the <i>Hub</i> can source more than 3A, it initiates a <i>VCONN Swap</i> to become the <i>VCONN Source</i> .	
11	<i>Hub</i> reads the e-marker to determine the cable's current carrying capability.	
12	<i>Hub</i> initiates a <i>Power Role Swap</i> to become the <i>Source</i>	
13	<i>Hub</i> sends a Source_Capabilities Message with the Unconstrained Power bit set and Maximum Current > 0.	
14	<i>Hub</i> and laptop establish an <i>Explicit Contract</i> with <i>Hub</i> as source.	
15		Laptop sends a Get_Sink_Cap Message to determine the current required by the <i>Hub</i> to support an <i>FRS</i> . If the <i>Hub</i> does not support <i>FRS</i> or the laptop cannot supply the required current, the laptop Ignores any <i>Fast Role Swap Requests</i> it might see.

Table E.2 Sequence for setup of a Fast Role Swap (Hub connected to laptop before Power Adapter)

Step #	Hub	Laptop
16		If the laptop has detected that it is connected via an <i>Active Cable</i> (or one that supports <i>Alternate Modes</i>) and/or that it can support an <i>FRS</i> , it initiates a <i>VCONN Swap</i> to make itself the <i>VCONN Source</i> . This removes a requirement that the <i>Hub</i> also hold up <i>VCONN</i> during the <i>FRS</i> .
17	The <i>Hub</i> and laptop are now ready to do a <i>Fast Role Swap</i> in case the power adapter gets removed.	

E.3 FRS Process

After the initial setup is completed and the laptop has determined both that the *Hub* can request *FRS* and that the laptop is able and willing to supply the requested current, the system is ready to support *FRS*. This section describes the sequence of events that take place during a *Fast Role Swap*. The following figures and tables do not cover the actions of the *Device Policy Manager* or the Policy Engine. Those actions occur orthogonally to the electrical events shown in this appendix. However, the diagrams do indicate the inputs/outputs where the DPM and Policy Engine interact with the electrical events:

- The laptop sends the **FR_Swap** Message to initiate the *FRS AMS* (see [Figure 7.43, "Transition Diagram for Fast Role Swap"](#)) within 15ms after the laptop detects the *Fast Role Swap Request* on *CC*.
- The laptop sends the final **PS_RDY** Message in the *FRS AMS* only after it is sourcing *V_{BUS}*.

Figure E.4 Slow *V_{BUS}* discharge after *FR_Swap* message is sent

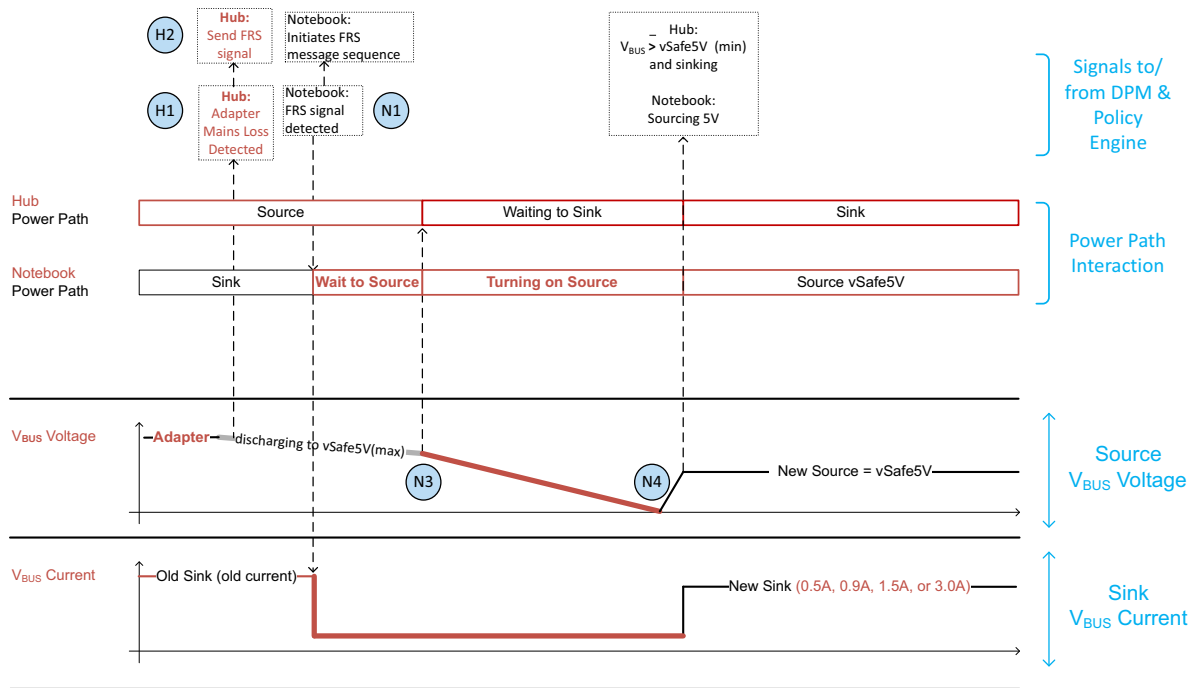


Table E.3 Sequence for slow VBUS discharge (it discharges after FR_Swap message is sent)

Step #	Hub	Laptop
1	The power adapter's <i>AC Supply</i> power is lost.	
2	<i>Hub</i> detects the power adapter disconnect (H1) as quickly as possible.	
3	<i>Hub</i> sends <i>Fast Role Swap Request</i> on <i>CC</i> (H2) and starts monitoring V_{HubVB} (H3). <i>Hub</i> also starts a <i>tSnkFRSwap</i> timer after the FRS signal begins and <i>VBUS</i> has fallen below <i>vSafe5V</i> (min).	
4		Laptop detects <i>Fast Role Swap Request</i> on <i>CC</i> (N1) that triggers sending of the <i>FR_Swap Message</i> . This can happen at any point in the following steps so long as it is within 15 ms (<i>tFRSwapInit</i>).
5		Laptop opens the sinking switch (N2), as quickly as possible to minimize power drained from <i>Hub</i> after the <i>Fast Role Swap Request</i> .
6		Laptop begins monitoring <i>VBUS</i> (N3) to know when to turn the laptop into a <i>Source</i> .
7	<i>Hub</i> opens the sourcing switch (H4) while $V_{HubVB} > 5.5V$ (after the <i>Fast Role Swap Request</i> is sent). However, the sourcing switch (H4) must be kept closed until V_{HubVB} is as close to 5.5V as possible. It is important for the <i>Hub</i> to open its sourcing switch (H4) before the laptop's sourcing switch (N4) gets closed to minimize inrush current.	
8	<i>Hub</i> closes the switch (H5) to use the hold-up capacitor to supply <i>VBUS</i> to the peripheral(s). Systems with a holding cap permanently in place do not need the switch (H5). <i>Hub</i> does not draw more than <i>iSnkStdby</i> from <i>VBUS</i> , until the <i>tSnkFRSwap</i> timer expires.	
9		Laptop detects $V_{BUS} < V_{NbVB}$ (N1) before closing the sourcing switch (N4) when V_{NbVB} is as close as possible to 5.5V. This minimizes the time when <i>VBUS</i> is not sourced.
10		Laptop closes sourcing switch (N4). When this occurs the <i>Hub</i> 's input capacitance on <i>VBUS</i> will be less than 10 μ F (<i>cSnkBulk</i>).
11	<i>Hub</i> 's <i>tSnkFRSwap</i> timer expires (H6).	
12	<i>Hub</i> draws up to the current it <i>Advertised</i> in the <i>Fast Role Swap required USB Type-C Current</i> field of its <i>Sink_Capabilities Message</i> .	
13	<i>Hubs</i> with (H5) will open (H5) and remove the Hold-Up capacitor.	

Figure E.5 VBUS discharges quickly before FR_Swap message is sent after adapter disconnected

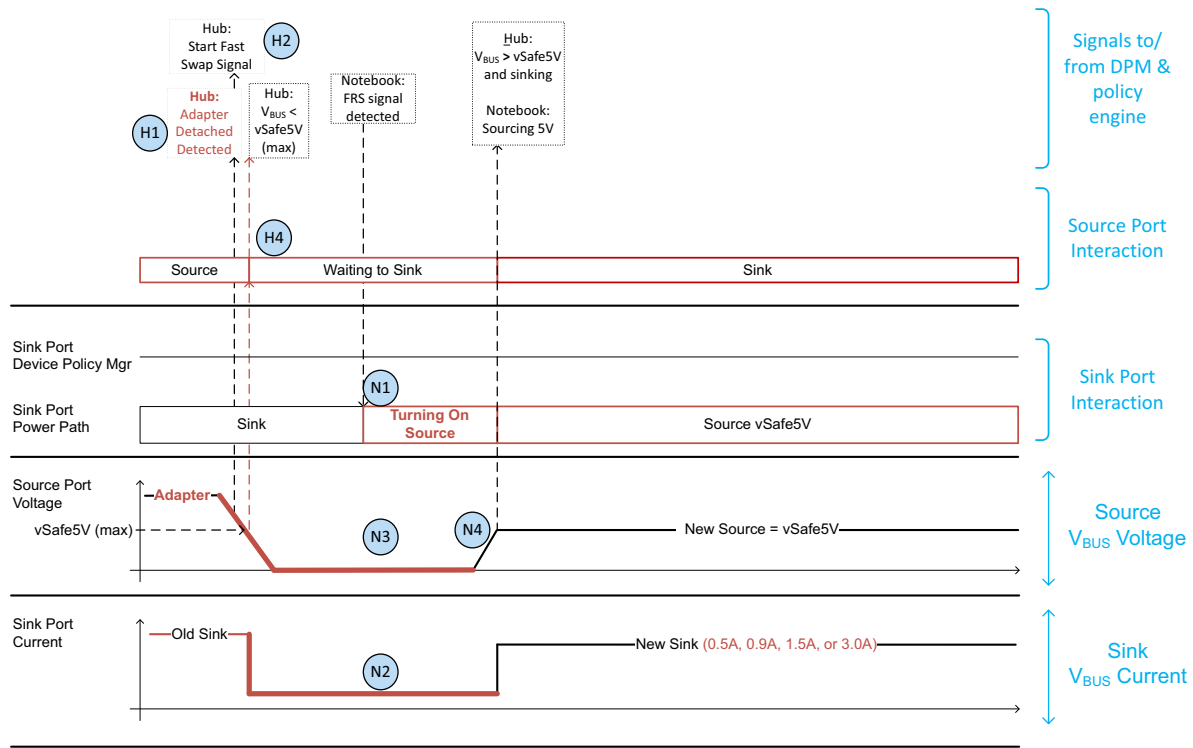


Table E.4 VBUS discharges quickly after adapter disconnected

Step #	Hub	Laptop
1	The power adapter is <i>Detached</i> from the <i>Hub</i> .	
2	<i>Hub</i> detects power adapter disconnect (H1)-causing V_{HubVB} to drop below 5.5V very rapidly.	
3	<i>Hub</i> sends <i>Fast Role Swap Request</i> on CC (H2) and starts monitoring V_{HubVB} (H3). <i>Hub</i> opens sourcing switch (H4). <i>Hub</i> also starts a <i>tSnkFRSwap</i> timer.	
4	<i>Hub</i> closes the switch (H5) to use the hold-up capacitor to supply $VBUS$ to the peripheral(s). Systems with a holding cap permanently in place do not need the switch (H5). <i>Hub</i> does not draw more than <i>iSnkStdby</i> from $VBUS$, until the <i>tSnkFRSwap</i> timer expires.	
5		Laptop detects <i>Fast Role Swap Request</i> on CC (N1) that triggers sending of the <i>FR_Swap Message</i> . This can happen at any point in the following steps so long as it is within 15 ms (<i>tFRSwapInit</i>).
6		Laptop opens the sinking switch (N2), as quickly as possible to minimize power drained from <i>Hub</i> after the <i>Fast Role Swap Request</i> .
7		Laptop begin monitoring $VBUS$ (N3) to know when to turn the laptop into a <i>Source</i>

Table E.4 VBUS discharges quickly after adapter disconnected

Step #	Hub	Laptop
8		Laptop detects $V_{BUS} < V_{NbVB}$ (N3).
9		Laptop closes sourcing switch (N4). When this occurs the <i>Hub's</i> input capacitance on V_{BUS} will be less than 10 μF (<i>cSnkBulk</i>).
10	<i>Hub's tSnkFRSwap</i> timer expires (H6).	
11	<i>Hub</i> draws up to the current it <i>Advertised</i> in the <i>Fast Role Swap required USB Type-C Current</i> field of its <i>Sink_Capabilities</i> Message.	
12	<i>Hubs</i> with (H5) will open (H5) and remove the Hold-Up capacitor.	