

[MS-RDPEDYC]: Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft [Open Specification Promise](#) or the [Community Promise](#). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	1.0.1	Minor	Editorial fixes only.
07/20/2007	1.0.2	Editorial	Revised and edited the technical content.
08/10/2007	1.0.3	Editorial	Revised and edited the technical content.
09/28/2007	1.0.4	Editorial	Revised and edited the technical content.
10/23/2007	1.0.5	Minor	Revised a figure; revised size of cmd field in several packets.
11/30/2007	1.1	Minor	Updated the technical content.
01/25/2008	1.1.1	Editorial	Revised and edited the technical content.
03/14/2008	1.1.2	Editorial	Revised and edited the technical content.
05/16/2008	1.1.3	Editorial	Revised and edited the technical content.
06/20/2008	1.2	Minor	Updated the technical content.
07/25/2008	1.2.1	Editorial	Revised and edited the technical content.
08/29/2008	1.2.2	Editorial	Revised and edited the technical content.
10/24/2008	1.2.3	Editorial	Revised and edited the technical content.
12/05/2008	2.0	Major	Updated and revised the technical content.
01/16/2009	2.1	Minor	Updated the technical content.
02/27/2009	2.1.1	Editorial	Revised and edited the technical content.
04/10/2009	2.1.2	Editorial	Revised and edited the technical content.
05/22/2009	3.0	Major	Updated and revised the technical content.
07/02/2009	3.0.1	Editorial	Revised and edited the technical content.
08/14/2009	3.0.2	Editorial	Revised and edited the technical content.
09/25/2009	3.0.3	Editorial	Revised and edited the technical content.
11/06/2009	3.0.4	Editorial	Revised and edited the technical content.
12/18/2009	4.0	Major	Updated and revised the technical content.
01/29/2010	5.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
03/12/2010	6.0	Major	Updated and revised the technical content.
04/23/2010	6.0.1	Editorial	Revised and edited the technical content.
06/04/2010	7.0	Major	Updated and revised the technical content.
07/16/2010	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	7.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	8.0	Major	Significantly changed the technical content.
03/25/2011	9.0	Major	Significantly changed the technical content.
05/06/2011	9.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	9.1	Minor	Clarified the meaning of the technical content.
09/23/2011	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	10.0	Major	Significantly changed the technical content.
03/30/2012	10.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	11.0	Major	Significantly changed the technical content.
10/25/2012	11.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	11.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	12.0	Major	Significantly changed the technical content.
11/14/2013	12.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/13/2014	13.0	Major	Significantly changed the technical content.

Contents

1 Introduction	7
1.1 Glossary	7
1.2 References.....	8
1.2.1 Normative References.....	8
1.2.2 Informative References	8
1.3 Overview	8
1.3.1 Encapsulation of DVC Traffic	9
1.3.1.1 Encapsulation in the DRDYNVC Static Virtual Channel.....	9
1.3.1.2 Encapsulation in a Multitransport Tunnel Message.....	9
1.3.2 DVC Setup.....	9
1.3.3 Message Flows	10
1.3.3.1 Opening a DVC.....	10
1.3.3.2 Sending and Receiving Data.....	11
1.3.3.2.1 Sending Data.....	12
1.3.3.2.2 Receiving Data.....	12
1.3.3.3 Closing a DVC	13
1.4 Relationship to Other Protocols.....	13
1.5 Prerequisites/Preconditions	13
1.6 Applicability Statement.....	13
1.7 Versioning and Capability Negotiation.....	13
1.8 Vendor-Extensible Fields.....	14
1.9 Standards Assignments	14
2 Messages	15
2.1 Transport.....	15
2.2 Message Syntax	15
2.2.1 Initializing DVCs.....	16
2.2.1.1 DVC Capabilities Request PDU	17
2.2.1.1.1 Version 1 (DYNVC_CAPS_VERSION1)	17
2.2.1.1.2 Version 2 (DYNVC_CAPS_VERSION2)	17
2.2.1.1.3 Version 3 (DYNVC_CAPS_VERSION3)	19
2.2.1.2 DVC Capabilities Response PDU (DYNVC_CAPS_RSP)	20
2.2.2 Opening a DVC	20
2.2.2.1 DVC Create Request PDU (DYNVC_CREATE_REQ)	20
2.2.2.2 DVC Create Response PDU (DYNVC_CREATE_RSP)	21
2.2.3 Sending and Receiving Data	22
2.2.3.1 DVC Data First PDU (DYNVC_DATA_FIRST).....	22
2.2.3.2 DVC Data PDU (DYNVC_DATA)	23
2.2.4 Closing a DVC (DYNVC_CLOSE).....	24
3 Protocol Details	26
3.1 Common Details	26
3.1.1 Abstract Data Model	27
3.1.2 Timers	28
3.1.3 Initialization	28
3.1.4 Higher-Layer Triggered Events.....	28
3.1.5 Message Processing Events and Sequencing Rules.....	28
3.1.5.1 Sending Data	29
3.1.5.1.1 DVC Data First (DYNVC_DATA_FIRST)	29
3.1.5.1.2 DVC Data (DYNVC_DATA)	29

3.1.5.2	Receiving Data	29
3.1.5.2.1	DVC Data First (DYNVC_DATA_FIRST)	29
3.1.5.2.2	DVC Data (DYNVC_DATA)	29
3.1.5.2.3	Reassembly of Fragmented Virtual Channel Data	29
3.1.5.2.4	Processing Packet Errors	30
3.1.6	Timer Events	30
3.1.7	Other Local Events	30
3.2	Client Details.....	30
3.2.1	Abstract Data Model	30
3.2.2	Timers	30
3.2.3	Initialization	30
3.2.3.1	DVC Client Manager Initialization.....	30
3.2.3.1.1	Version Level 1 (DYNVC_CAPS_VERSION1)	31
3.2.3.1.2	Version Level 2 (DYNVC_CAPS_VERSION2)	31
3.2.3.1.3	Version Level 3 (DYNVC_CAPS_VERSION3)	31
3.2.3.1.4	Capabilities Response (DYNVC_CAPS_RSP)	31
3.2.3.2	DVC Initialization.....	31
3.2.3.2.1	DVC Create Response (DYNVC_CREATE_RSP).....	31
3.2.4	Higher-Layer Triggered Events.....	32
3.2.5	Message Processing Events and Sequencing Rules.....	32
3.2.5.1	Sending and Receiving Data.....	32
3.2.5.2	Closing a DVC (DYNVC_CLOSE)	32
3.2.6	Timer Events	32
3.2.7	Other Local Events	32
3.3	Server Details	32
3.3.1	Abstract Data Model	32
3.3.2	Timers	32
3.3.3	Initialization	33
3.3.3.1	DVC Server Manager Initialization.....	33
3.3.3.1.1	Version Level 1 (DYNVC_CAPS_VERSION1)	33
3.3.3.1.2	Version Level 2 (DYNVC_CAPS_VERSION2)	33
3.3.3.1.3	Version Level 3 (DYNVC_CAPS_VERSION3)	33
3.3.3.1.4	Capabilities Response (DYNVC_CAPS_RSP)	34
3.3.3.2	DVC Initialization.....	34
3.3.4	Higher-Layer Triggered Events.....	34
3.3.5	Message Processing Events and Sequencing Rules.....	34
3.3.5.1	Sending and Receiving Data.....	34
3.3.5.2	Closing a DVC (DYNVC_CLOSE)	35
3.3.6	Timer Events	35
3.3.7	Other Local Events	35
4	Protocol Examples.....	36
4.1	Annotated Initializing DVCs.....	36
4.1.1	DVC Capabilities Request (Version2) PDU	36
4.1.2	DVC Capabilities Response PDU.....	36
4.2	Annotated Opening a DVC.....	37
4.2.1	DVC Create Request PDU	37
4.2.2	DVC Create Response PDU	37
4.3	Annotated Sending and Receiving Data	37
4.3.1	DVC Data First PDU	37
4.3.2	DVC Data PDU	40
4.4	Annotated Closing a DVC	42
4.4.1	DVC Close PDU	42

5 Security	43
5.1 Security Considerations for Implementers.....	43
5.2 Index of Security Parameters	43
6 Appendix A: Product Behavior	44
7 Change Tracking	45
8 Index	47

1 Introduction

The Remote Desktop Protocol: Dynamic Virtual Channel Extension is an extension and refinement of the **virtual channel** protocol, as specified in [\[MS-RDPBCGR\]](#). It supports features such as classes of priority (that may be used to implement bandwidth allocation) and individually connected endpoints using **dynamic virtual channel (DVC) listeners**.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

Windows-1252 character set

The following terms are specific to this document:

ANSI character: An 8-bit **Windows-1252 character set** unit.

data message (or message): Data exchanged between an application running on a **terminal services (TS)** server and a **dynamic virtual channel (DVC) listeners** running on a **TS** client. The maximum length of a **data message** is $2^{32} - 1$ bytes.

dynamic virtual channel (DVC): A transport used for communication between an RDP client and server over a main data connection.

dynamic virtual channel (DVC) listener (or listener): A named endpoint registered at the **TS** client during initialization of a **DVC**. **DVC listeners** are service providers to the applications that run on a **TS** server.

dynamic virtual channel (DVC) manager: An application that runs on the **TS** servers and clients. They manage the initialization, creation, and closing of **DVCs**. They are responsible for maintaining established channels and for transferring **messages** between the applications on the **TS** servers and the **DVC listeners** that run on the **TS** clients.

priority class: The priority of a group of channels. Channels of a higher priority class will typically be allotted a larger proportion of available bandwidth than those of a lower class.

static virtual channel: The Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension is designed to operate over static **virtual channels**, as specified in [\[MS-RDPBCGR\]](#), using the acronym DRDYNVC. The Remote Desktop Protocol (RDP) layer manages the creation, setup, and data transmission over the **virtual channel**.

terminal services (TS): The capability to host multiple, simultaneous client sessions on Windows servers. Remote users establish a session on a machine, log in, and run applications on a server. The server transmits the graphical user interface (GUI) of the program to the client. The client then returns keyboard and mouse clicks to be processed by the server.

virtual channel: A communication channel available in a **TS** server session between applications running at the server and applications running on the **TS** client.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as specified in [\[RFC2119\]](#). All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

A reference marked "(Archived)" means that the reference document was either retired and is no longer being maintained or was replaced with a new document that provides current implementation details. We archive our documents online [\[Windows Protocol\]](#).

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[MS-DTYP] Microsoft Corporation, "[Windows Data Types](#)".

[MS-ERREF] Microsoft Corporation, "[Windows Error Codes](#)".

[MS-RDPBCGR] Microsoft Corporation, "[Remote Desktop Protocol: Basic Connectivity and Graphics Remoting](#)".

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/rfc/rfc2119.txt>

1.2.2 Informative References

None.

1.3 Overview

The Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension implements a generic connection-oriented communication channel on top of the virtual channel protocol. A **DVC** is established over an existing **static virtual channel**. A static virtual channel session is a typical client/server relationship.

A DVC consists of two endpoints logically connected over a network. One endpoint is an application running on a **terminal services (TS)** server, and the other endpoint is an application running on a TS client.

DVCs are created and maintained by **DVC managers**. There is a DVC manager running on both the TS server and the TS client. The DVC server manager is responsible for initializing the DVC environment and for creating individual DVCs. The DVC client manager is responsible for creating and maintaining connections to client-side DVC manager applications.

After the DVC managers are initialized, the DVC server manager can create individual DVCs. These channels are used to exchange **messages** between applications running on the TS server and DVC listeners running on the TS client. Sending and receiving messages is symmetrical between the client and server, and either side can initiate sending a message.

1.3.1 Encapsulation of DVC Traffic

If a multitransport connection ([\[MS-RDPEMT\]](#) section 1.3) is associated with a given RDP connection, the DVC PDUs, specified in section 2.2, can be embedded inside either the dedicated DRDYNVC static virtual channel, or inside a Tunnel Data PDU ([\[MS-RDPEMT\]](#) section 2.2.2.3). If a multitransport connection is not present, then the DVC PDUs are encapsulated inside the dedicated DRDYNVC static virtual channel.

1.3.1.1 Encapsulation in the DRDYNVC Static Virtual Channel

The following diagram illustrates the wire-level encapsulation when a DVC is embedded inside the dedicated static virtual channel named DRDYNVC.

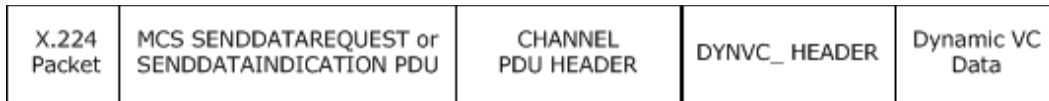


Figure 1: Static virtual channel objects

This is a Windows implementation detail and does not limit the definition and the description of the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension. Any transport that has similar characteristics can be used to support a DVC implementation. The Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension makes use of the following features of a static virtual channel:

- Capability to indicate the reception of a complete message to the DVC handler.
- Capability to support a minimum message size that is sufficient for the complete reception of the PDUs used for version negotiation and channel open/close functionality.

1.3.1.2 Encapsulation in a Multitransport Tunnel Message

The following diagram illustrates the wire-level encapsulation when a DVC is embedded inside a multitransport connection tunnel ([\[MS-RDPEMT\]](#) sections 1.3 and 1.4).



Figure 2: Encapsulation inside a multitransport connection tunnel

1.3.2 DVC Setup

The following diagram illustrates the sequence of operations involved in initializing the client and server environments.

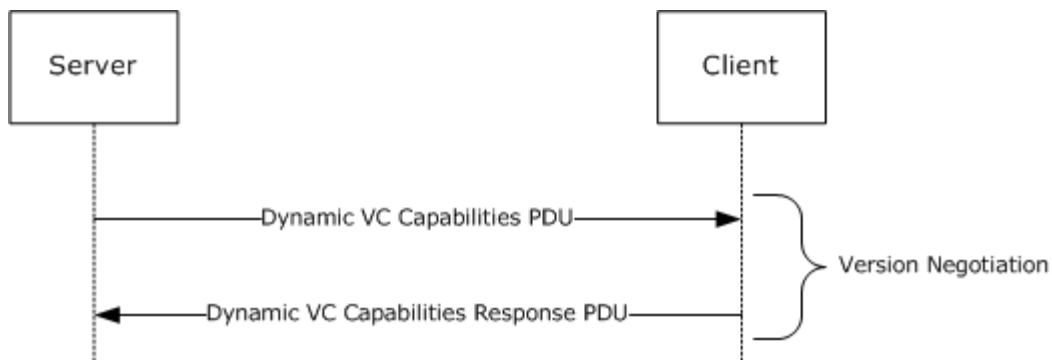


Figure 3: DVC initialization sequence

The initialization is performed immediately following the establishment of a static virtual channel session, as specified in [\[MS-RDPBCGR\]](#) section 1.3.3.

The initialization is performed once per connection. At startup and initialization, a DVC server manager performs a version negotiation with a DVC client manager over the existing static virtual channel.

The client and server initialize their environments by exchanging a capability message. The DVC server manager sends a capabilities protocol data unit (PDU) that indicates the maximum supported version level as well as any capability information that is relevant for the supported version. The capability information describes the features supported by the server.

The DVC client manager responds with a capabilities response PDU that states the maximum version level that it supports. The server should adjust the protocol features to match the client capabilities. After this negotiation, the DVC server manager and DVC client manager are ready to establish individual DVCs.

1.3.3 Message Flows

1.3.3.1 Opening a DVC

The following diagram illustrates the sequence of operations involved in the creation of a DVC.

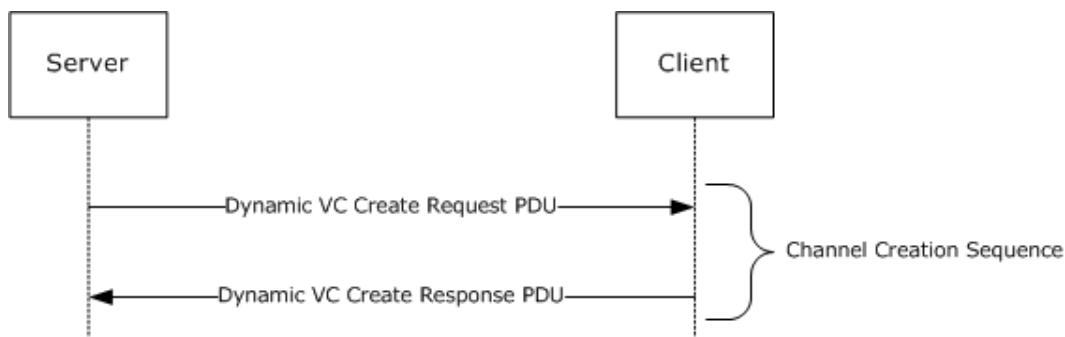


Figure 4: DVC open sequence

A DVC consists of two endpoints logically connected over a network. One endpoint is an application running on a TS server, and the other endpoint is an application running on a TS client. The

applications running on the TS client are referred to as DVC listeners. These listeners are service providers to the applications running on the TS server.

Channels are established by the DVC managers exchanging Create Request and Create Response PDUs. Channels are created by a DVC server manager in response to a channel-create request by an application. When an application makes a request to a DVC server manager to create a channel, the server generates a channel identifier (that is, a unique number for the requested session), and sends this identifier (and the listener name the application is requesting a connection to) in a Create Request PDU to the DVC client manager. The DVC client manager locates the requested listener, and the listener creates a DVC using the **ChannelId**. The DVC client manager binds the endpoint to the **ChannelId**. The client then sends a Create Response message to the server indicating the endpoint creation status. If the creation is successful, the DVC server manager indicates to the application that the session is established and is ready for sending and receiving data. The client and server maintain the endpoints for the life of the channel.

When a multitransport connection ([\[MS-RDPENMT\]](#) section 1.3) is negotiated for a given RDP connection, the DVC server manager can establish a particular DVC from one of the connected transports (TCP, UDP-R, or UDP-L) based on preferences specified by the application endpoint. For example, an application can request that the DVC be encapsulated in a TCP transport for a feature that requires full-reliability but can be latency-tolerant, such as printer redirection. An application can also request that the DVC be encapsulated in a UDP-L transport for a feature that is loss-tolerant but latency-sensitive, such as audio output redirection.

The server DVC manager sends the Create Request PDU over the selected transport, and the client responds by sending the Create Response PDU back to the server over the same transport.

1.3.3.2 Sending and Receiving Data

The maximum size of a message that a sender can pass to a DVC manager is $2^{32}-1$ bytes (see section [3.1.5.2.3](#)). The Remote Desktop Protocol: Dynamic Virtual Channel Extension specifies a maximum PDU size of 1,600 bytes. A data PDU consists of a header and message data. The DVC manager is responsible for fragmenting and reassembling large messages. This is provided as a service to the sender and receiver.

The sending and receiving of messages is symmetrical between the client and server, and either side can initiate sending a message. Depending on the size of the message being sent, there are two different message sequences for sending and receiving data, as illustrated in the following diagram.

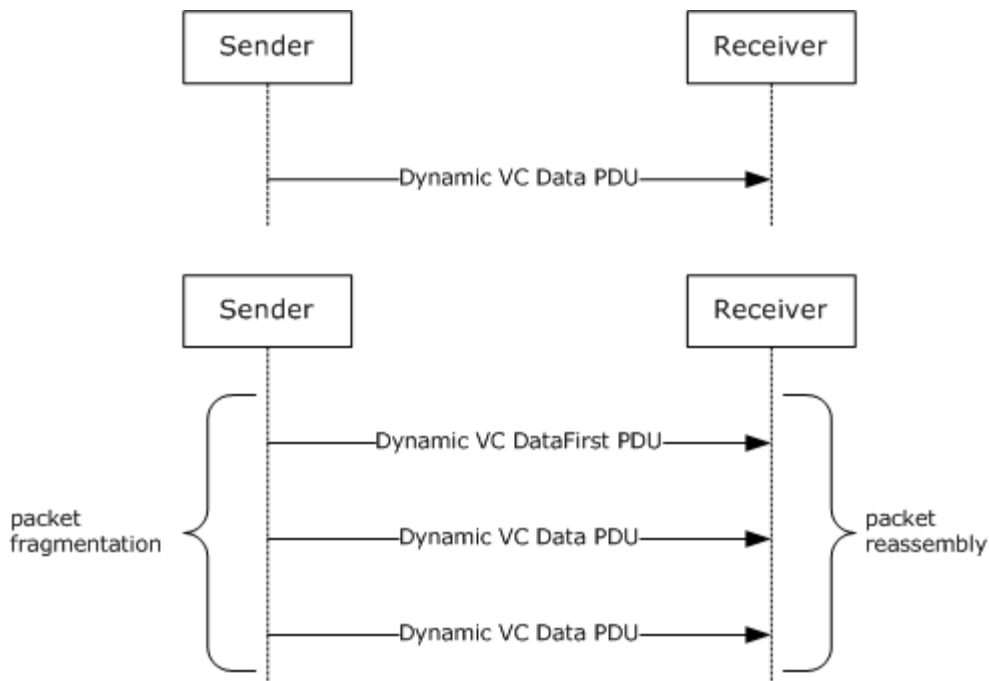


Figure 5: Send data sequence via DVC PDUs

1.3.3.2.1 Sending Data

If the sender makes a request to send a message that has a data size of no more than 1590 bytes, then a single PDU is sent that contains the message and a header with the **Cmd** field set to indicate that the PDU type is Data.

If the sender makes a request to send a message that has a data size of more than 1590 bytes, then the DVC manager fragments the message into blocks and may send multiple PDUs. The first PDU contains the first fragment of message data and a header with the **Cmd** field set to indicate that the PDU type is Data First and the **Length** field set to the total length of the message the sender is sending. Subsequent PDUs of type Data are sent until the entire message is transmitted.

The receiver does not acknowledge receipt of the data.

1.3.3.2.2 Receiving Data

If a message has been fragmented, the first data PDU received will be of type Data First. If the message has not been fragmented, the first and only PDU for this message will be of type Data.

When a DVC manager receives a Data First PDU, it saves the message data and continues receiving Data PDUs until all the data is received. The DVC manager then reassembles the data and passes the data to the receiver associated with this channel.

When a DVC manager receives a Data PDU that has not been preceded by a Data First PDU, it passes the message data directly to the receiver without any additional processing.

The DVC manager does not acknowledge receipt of the data.

1.3.3.3 Closing a DVC

Either an application running on the TS server or a listener running on the TS client can request that a channel be closed. The following diagram illustrates the sequence of operations involved in closing a DVC.

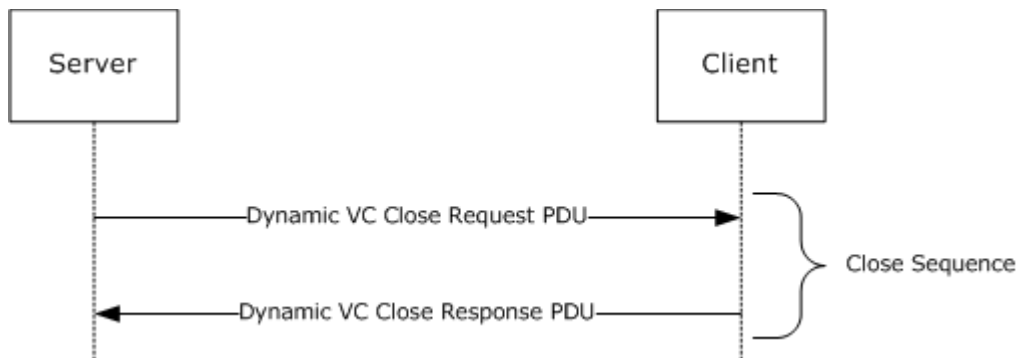


Figure 6: DVC close sequence

When the DVC server manager initiates closing a channel, it sends a Close Request PDU that specifies the **ChannelId** to the DVC client manager. The client responds with a Close Response PDU that specifies the **ChannelId**.

When a client initiates a channel-close, it sends an unsolicited Close Response PDU that specifies the **ChannelId** to the server. The server does not respond to the Client Close Response PDU.

1.4 Relationship to Other Protocols

The Remote Desktop Protocol: Dynamic Virtual Channel Extension is embedded in a static virtual channel transport, as specified in [\[MS-RDPBCGR\]](#).

1.5 Prerequisites/Preconditions

The Remote Desktop Protocol: Dynamic Virtual Channel Extension operates only after the static virtual channel transport (as specified in [\[MS-RDPBCGR\]](#)) is fully established. If the static virtual channel transport is terminated, no other communication over the Remote Desktop Protocol: Dynamic Virtual Channel Extension occurs.

1.6 Applicability Statement

The Remote Desktop Protocol: Dynamic Virtual Channel Extension is designed to be run within the context of an RDP virtual channel established between a client and a server. The Remote Desktop Protocol: Dynamic Virtual Channel Extension is applicable when creating applications such as Plug and Play device redirection and media infrastructure layer composition engine commands.

1.7 Versioning and Capability Negotiation

The version of the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension is negotiated by a DVC client manager in response to a Capabilities PDU sent by a DVC server manager. The server indicates the maximum capability it supports, and the client responds with a Capabilities Response PDU that indicates the maximum capability that it can support.

There are two versions of the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension.

- Version 1: The first version of the protocol consists of initialization, closing, and the sending of data over DVCs.
- Version 2: The second version introduced the capability to specify priority classes for virtual channels in order to allocate different bandwidth to different classes of virtual channels.

Implementations can support either version 1 or both version 1 and version 2 of the protocol. The negotiation of the protocol between the server and the client is described in section [2.2.1](#).

1.8 Vendor-Extensible Fields

This protocol uses NTSTATUS values as defined in [\[MS-ERREF\]](#) section 2.3. Vendors are free to choose their own values for this field, as long as the C bit (0x20000000) is set, indicating it is a customer code.

1.9 Standards Assignments

The Remote Desktop Protocol: Dynamic Virtual Channel Extension does not use any standards assignments.

2 Messages

The following sections specify how Remote Desktop Protocol: Dynamic Virtual Channel Extension messages are encapsulated on the wire and common data types.

This protocol references commonly used data types as defined in [\[MS-DTYP\]](#).

2.1 Transport

Remote Desktop Protocol: Dynamic Virtual Channel Extension messages are passed between a DVC manager on a server and a DVC manager on a client, and are transmitted over one of the following underlying transport mechanisms:

- A static virtual channel ([\[MS-RDPBCGR\]](#) sections [1.3.3](#) and [2.2.6](#)), where the name of the channel is the **ANSI character** string "DRDYNV" and bulk compression is enabled.
- A lossy or reliable UDP multitransport connection ([\[MS-RDPENT\]](#) sections [1.3](#) and [2.2.2.3](#)).

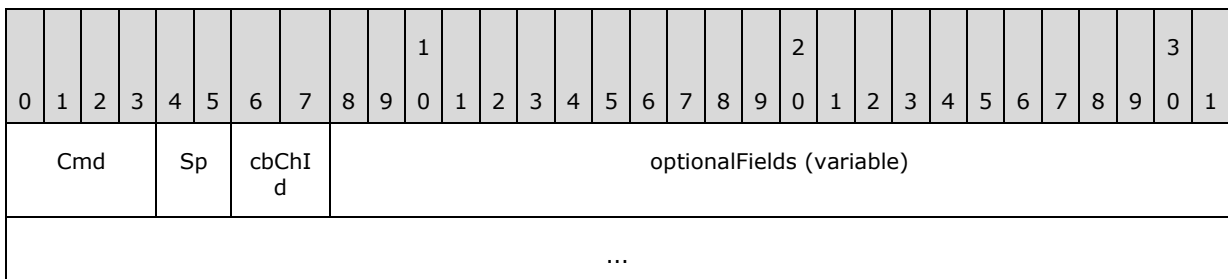
The Remote Desktop Protocol: Dynamic Virtual Channel Extension does not establish any transport connections.

2.2 Message Syntax

The Remote Desktop Protocol: Dynamic Virtual Channel Extension consists of the following five types of messages exchanged between the server and the client.

- Capability Negotiation message
- Channel Open message
- Channel DataFirst message
- Channel Data message
- Channel Close message

Each PDU has the same 1-byte header with the **optionalFields** field following it. The **cbChId** and **Cmd** fields are common to all PDUs. The data following the PDU header depends on the type of the message and is addressed in the following sections.



Cmd (4 bits): Indicates the PDU type and MUST be set to one of the following values.

Value	Meaning
0x01	The message contained in the optionalFields field is a Create Request PDU (section

Value	Meaning
	2.2.2.1) or a Create Response PDU (section 2.2.2.2) .
0x02	The message contained in the optionalFields field is a Data First PDU (section 2.2.3.1) .
0x03	The message contained in the optionalFields field is a Data PDU (section 2.2.3.2) .
0x04	The message contained in the optionalFields field is a Close Request PDU (section 2.2.4) or a Close Response PDU (section 2.2.4).
0x05	The message contained in the optionalFields field is a Capability Request PDU (section 2.2.1.1) or a Capabilities Response PDU (section 2.2.1.2) .

Sp (2 bits): The value and meaning depend on the **Cmd** field.

cbChId (2 bits): Indicates the length of the **ChannelId** field.

Value	Meaning
0x00	The ChannelId is 1 byte wide.
0x01	The ChannelId is 2 bytes wide.
0x02	The ChannelId is 4 bytes wide.
0x03	The ChannelId is 4 bytes wide.

optionalFields (variable): The data following the message header depends on the type of the message and is addressed in the following sections.

2.2.1 Initializing DVCs

Capabilities PDUs are exchanged to negotiate the version level of the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension that is supported. Three different Capabilities PDUs are used to negotiate version-level support.

- [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#) PDU is sent by a DVC server manager to indicate it supports version 1 of the protocol.
- [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#) PDU is sent by a DVC server manager to indicate it supports version 2 of the protocol.
- [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU is sent by a DVC server manager to indicate it supports version 3 of the protocol.
- [DYNVC_CAPS_RSP \(section 2.2.1.2\)](#) PDU is sent by a DVC client manager to acknowledge the version level it supports.

A DVC server manager initializes a DVC environment by sending a DYNVC_CAPS_VERSION1 (section 2.2.1.1.1), a DYNVC_CAPS_VERSION2 (section 2.2.1.1.2), or a DYNVC_CAPS_VERSION3 (section 2.2.1.1.3) PDU to the DVC client manager to indicate the highest version level supported by the server. The client **MUST** respond with a DYNVC_CAPS_RSP (section 2.2.1.2) PDU that indicates the highest version level supported by the client.

The DVC server manager **MUST** send a Capabilities message prior to creating a DVC and wait for a response from the client. This happens just once; if capability exchange has already been

completed, the channel creation continues. The DVC client manager MUST reply with a DYNVC_CAPS_RSP (section 2.2.1.2) PDU as soon as it receives the server request.

2.2.1.1 DVC Capabilities Request PDU

2.2.1.1.1 Version 1 (DYNVC_CAPS_VERSION1)

The DYNVC_CAPS_VERSION1 PDU is sent by the DVC server manager to indicate that it supports version 1 of the Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension. [<1>](#)

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		Pad								Version															

Cmd (4 bits): MUST be set to 0x05 (Capabilities).

Sp (2 bits): Unused. SHOULD be initialized to 0x00. [<2>](#)

cbChId (2 bits): Unused. MUST be set to 0x00.

Pad (1 byte): An 8-bit unsigned integer. Unused. MUST be set to 0x00.

Version (2 bytes): A 16-bit unsigned integer. MUST be set to 0x0001.

2.2.1.1.2 Version 2 (DYNVC_CAPS_VERSION2)

The DYNVC_CAPS_VERSION2 PDU is sent by the DVC server manager to indicate that it supports version 2 of the Remote Desktop Protocol: Dynamic Virtual Channel Extension. [<3>](#)

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		Pad								Version															
PriorityCharge0																PriorityCharge1															
PriorityCharge2																PriorityCharge3															

Cmd (4 bits): MUST be set to 0x05 (Capabilities).

Sp (2 bits): Unused. SHOULD be set to 0x00. [<4>](#)

cbChId (2 bits): Unused. MUST be set to 0x00.

Pad (1 byte): An 8-bit unsigned integer. Unused. MUST be set to 0x00.

Version (2 bytes): A 16-bit unsigned integer. MUST be set to 0x0002.

PriorityCharge0 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each **priority class**, in accordance with the following algorithm.

PriorityCharge1 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the following algorithm.

PriorityCharge2 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the following algorithm.

PriorityCharge3 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the following algorithm.

The **PriorityCharge** fields determine how much bandwidth is allocated for each priority class. The percentage is calculated using the following formula.

```
Base = PriorityCharge0 * PriorityCharge1 *
      PriorityCharge2 * PriorityCharge3 /
      (PriorityCharge1 * PriorityCharge2 *
      PriorityCharge3 + PriorityCharge0 *
      PriorityCharge2 * PriorityCharge3 +
      PriorityCharge0 * PriorityCharge1 * PriorityCharge3 +
      PriorityCharge0 * PriorityCharge1 *
      PriorityCharge2)

BandwidthPriority0 = Base / PriorityCharge0
BandwidthPriority1 = Base / PriorityCharge1
BandwidthPriority2 = Base / PriorityCharge2
BandwidthPriority3 = Base / PriorityCharge3
```

Where BandwidthPriorityX is a number between 0 and 1, and the total sum of all BandwidthPriorityX values is equal to 1. If PriorityChargeX is set to 0, the priority formula is not used and the data is sent immediately; remaining bandwidth is shared among channels with non-zero PriorityChargeX values, as described in the formula.

To calculate priority charges from given priorities the formula is as follows.

```
PriorityCharge0 = 65536 / (BandwidthPriority0 * 100)
PriorityCharge1 = 65536 / (BandwidthPriority1 * 100)
PriorityCharge2 = 65536 / (BandwidthPriority2 * 100)
PriorityCharge3 = 65536 / (BandwidthPriority3 * 100)
```

Where BandwidthPriorityX is a number between 0 and 1, and the total sum of all BandwidthPriorityX values is equal to 1.

For example, to have distribution for priority 0 to 3 be 70%, 20%, 7%, and 3%, the priority charges numbers are as follows.

```
PriorityCharge0 = 65536 / (0.70*100) = 936
PriorityCharge1 = 65536 / (0.20*100) = 3276
PriorityCharge2 = 65536 / (0.07*100) = 9362
PriorityCharge3 = 65536 / (0.03*100) = 21845
```

Calculating the priority from priority charges, as follows.

$$\text{Base} = 936 * 3276 * 9362 * 21845 / (3276 * 9362 * 21845 + 936 * 9362 * 21845 + 936 * 3276 * 21845 + 936 * 3276 * 9362) = 655$$

BandwidthPriority0 = 655/936 = 70%
 BandwidthPriority1 = 655/3276 = 20%
 BandwidthPriority2 = 655/9362 = 7%
 BandwidthPriority3 = 655/21845 = 3%

2.2.1.1.3 Version 3 (DYNVC_CAPS_VERSION3)

The DYNVC_CAPS_VERSION3 PDU is sent by the DVC server manager to indicate that it supports version 3 of the Remote Desktop Protocol: Dynamic Virtual Channel Extension. [<5>](#)

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		Pad								Version															
PriorityCharge0																PriorityCharge1															
PriorityCharge2																PriorityCharge3															

Cmd (4 bits): MUST be set to 0x05 (Capabilities).

Sp (2 bits): Unused. SHOULD be set to 0x00. [<6>](#)

cbChId (2 bits): Unused. MUST be set to 0x00.

Pad (1 byte): An 8-bit unsigned integer. Unused. MUST be set to 0x00.

Version (2 bytes): A 16-bit unsigned integer. MUST be set to 0x0003.

PriorityCharge0 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the algorithm specified in section [2.2.1.1.2](#).

PriorityCharge1 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the algorithm specified in section [2.2.1.1.2](#).

PriorityCharge2 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the algorithm specified in section [2.2.1.1.2](#).

PriorityCharge3 (2 bytes): A 16-bit unsigned integer. Specifies the amount of bandwidth that is allotted for each priority class, in accordance with the algorithm specified in section [2.2.1.1.2](#).

2.2.1.2 DVC Capabilities Response PDU (DYNVC_CAPS_RSP)

The DYNVC_CAPS_RSP (section 2.2.1.2) PDU is sent by the DVC client manager to the DVC server manager acknowledging the version level capabilities supported.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		Pad								Version															

Cmd (4 bits): MUST be set to 0x05 (Capabilities).

Sp (2 bits): Unused. MUST be set to 0x00.

cbChId (2 bits): Unused. MUST be set to 0x00.

Pad (1 byte): An 8-bit unsigned integer. Unused. MUST be set to 0x00.

Version (2 bytes): A 16-bit unsigned integer that indicates the protocol version level supported; MUST be set to the version level supported.

Value	Meaning
0x0001	Version level one is supported.
0x0002	Version level two is supported.
0x0003	Version level three is supported.

2.2.2 Opening a DVC

The DVC server manager initiates opening a DVC by exchanging Create PDUs with a DVC client manager. The server sends a [DYNVC_CREATE_REQ \(section 2.2.2.1\)](#) PDU to the client, and the client responds with a [DYNVC_CREATE_RSP \(section 2.2.2.2\)](#) PDU that indicates the status of the client endpoint creation.

2.2.2.1 DVC Create Request PDU (DYNVC_CREATE_REQ)

The DYNVC_CREATE_REQ (section 2.2.2.1) PDU is sent by the DVC server manager to the DVC client manager to request that a channel be opened.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Pri		cbChId		ChannelId (variable)																							
...																															
ChannelName (variable)																															

...

Cmd (4 bits): MUST be set to 0x01 (Create).

Pri (2 bits): Version 1 of the Remote Desktop Protocol: Dynamic Virtual Channel Extension (as specified in section [2.2.1.1.1](#)) does not support priority classes. The client SHOULD ignore this field.

In version 2 of the Remote Desktop Protocol: Dynamic Virtual Channel Extension, this field specifies the priority class for the channel that is being created, with the **Pri** field values 0, 1, 2, and 3 corresponding to PriorityCharge0, PriorityCharge1, PriorityCharge2, and PriorityCharge3, as specified in section [2.2.1.1.2](#). The method of determining priority class is the same for both client to server data and server to client data.

cbChId (2 bits): Indicates the length of the **ChannelId** field.

Value	Meaning
0x00	The ChannelId field length is 1 byte.
0x01	The ChannelId field length is 2 bytes.
0x02	The ChannelId field length is 4 bytes.
0x03	Invalid value.

ChannelId (variable): A variable-length 8-bit, 16-bit, or 32-bit unsigned integer. This is a server-generated identifier for the channel being created. The DVC server manager MUST ensure that this number is unique within a static virtual channel connection.

ChannelName (variable): A null-terminated ANSI encoded character string. The name of the listener on the TS client with which the TS server application is requesting that a channel be opened.

2.2.2.2 DVC Create Response PDU (DYNVC_CREATE_RSP)

The DYNVC_CREATE_RSP (section 2.2.2.2) PDU is sent by the DVC client manager to indicate the status of the client DVC create operation.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		ChannelId (variable)																							
...																															
CreationStatus																															

Cmd (4 bits): MUST be set to 0x01 (Create).

Sp (2 bits): Unused. SHOULD be initialized to 0x00.

cbChId (2 bits): Indicates the length of the **ChannelId** field.

Value	Meaning
0x00	The ChannelId field length is 1 byte.
0x01	The ChannelId field length is 2 bytes.
0x02	The ChannelId field length is 4 bytes.
0x03	Invalid value.

ChannelId (variable): A variable length 8-bit, 16-bit, or 32-bit unsigned integer. Set to the value of the **ChannelId** in the [DYNVC_CREATE_REQ \(section 2.2.2.1\)](#) PDU.

CreationStatus (4 bytes): A 32-bit, signed integer that specifies the HRESULT code that indicates success or failure of the client DVC creation. HRESULT codes are specified in [\[MS-ERREF\]](#) section 2.1. A zero or positive value indicates success; a negative value indicates failure.

2.2.3 Sending and Receiving Data

The maximum size of a message that a sender can pass to a DVC manager is $2^{32}-1$ bytes. The Remote Desktop Protocol: Dynamic Virtual Channel Extension specifies a maximum PDU size of 1,600 bytes. A data PDU consists of a header and message data. When a message that has a data size exceeding 1590 bytes is required to be sent by a DVC manager, the message MUST be sent using a [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#) structure, followed by an optional sequence of [DYNVC_DATA \(section 2.2.3.2\)](#) structures.

For an overview of sending and receiving data, see section [1.3.3.2](#).

2.2.3.1 DVC Data First PDU (DYNVC_DATA_FIRST)

The DYNVC_DATA_FIRST (section 2.2.3.1) PDU is used to send the first block of data of a fragmented message. It MUST be the first PDU sent when a message has been fragmented. The total length, in bytes, of the message to be sent is indicated in the **Length** field, and the data field contains the first block of the fragmented data.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Len		cbChId		ChannelId (variable)																							
...								Length (variable)																							
...								Data (variable)																							
...																															

Cmd (4 bits): This field MUST be set to 0x02 (Data First).

Len (2 bits): Indicates the length of the **Length** field.

Value	Meaning
0x0	Length field length is 1 byte.
0x1	Length field length is 2 bytes.
0x2	Length field length is 4 bytes.
0x3	Invalid value; MUST NOT be used.

cbChId (2 bits): Indicates the length of the **ChannelId** field.

Value	Meaning
0x0	ChannelId field length is 1 byte.
0x1	ChannelId field length is 2 bytes.
0x2	ChannelId field length is 4 bytes.
0x3	Invalid value.

ChannelId (variable): A variable-length 8-bit, 16-bit, or 32-bit unsigned integer. Set to the value of the **ChannelId** associated with the DVC on which the PDU is being sent.

Length (variable): A variable length 8-bit, 16-bit, or 32-bit unsigned integer. Set to total length of the message to be sent.

Data (variable): An array of bytes. The first block of data of a fragmented message. Message data is sent as 8-bit unsigned integers. The DVC header size is defined as the sum of the sizes of the **Cmd**, **Len**, **cbChId**, **ChannelId** and **Length** fields. The length of the data in the **Data** field is determined as follows:

- If the sum of the DVC header size and the value specified by the **Length** field is less than 1,600 bytes, then the actual data length equals the value specified by the **Length** field.
- If the sum of the DVC header size and the value specified by the **Length** field is equal to or larger than 1,600 bytes, then the actual data length equals 1,600 bytes minus the DVC header size.

2.2.3.2 DVC Data PDU (DYNVC_DATA)

The DYNVC_DATA (section 2.2.3.2) PDU is used to send both single messages and blocks of fragmented messages.

A single DYNVC_DATA (section 2.2.3.2) PDU is used to send a message when the total length of the message data is less than or equal to 1,590 bytes.

Multiple DYNVC_DATA (section 2.2.3.2) PDUs are used to send messages that have been fragmented and are sent subsequent to a [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#) PDU. DYNVC_DATA (section 2.2.3.2) PDUs are sent until the entire fragmented message has been sent.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		ChannelId (variable)																							
...																															
Data (variable)																															
...																															

Cmd (4 bits): MUST be set to 0x03 (Data).

Sp (2 bits): Unused. SHOULD be initialized to 0x00. <7>

cbChId (2 bits): Indicates the length of the **ChannelId** field.

Value	Meaning
0x00	ChannelId field length is 1 byte.
0x01	ChannelId field length is 2 bytes.
0x02	ChannelId field length is 4 bytes.
0x03	Invalid value.

ChannelId (variable): A variable-length 8-bit, 16-bit, or 32-bit unsigned integer. Set to the value of the **ChannelId** associated with the DVC upon which the PDU is being sent.

Data (variable): An array of bytes. Message data is sent as 8-bit unsigned integers. The maximum size of the array is 1,600 minus the length of the DYNVC_DATA header in bytes. The actual size of this field is the length of the packet after reassembly, as described in [\[MS-RDPBCGR\]](#) section 3.1.5.2.2.1, minus the space taken for **Cmd**, **Sp**, **cbChId**, and **ChannelId** fields.

2.2.4 Closing a DVC (DYNVC_CLOSE)

A DYNVC_CLOSE (section 2.2.4) PDU is sent by either a DVC server manager or a DVC client manager to close a DVC. A DYNVC_CLOSE (section 2.2.4) PDU is used for both a close request and a close response.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Cmd				Sp		cbChId		ChannelId (variable)																							
...																															

Cmd (4 bits): MUST be set to 0x04. (Close)

Sp (2 bits): Unused. SHOULD be initialized to 0x00.

cbChId (2 bits): Indicates the length of the **ChannelId** field.

Value	Meaning
0x00	ChannelId field length is 1 byte.
0x01	ChannelId field length is 2 bytes.
0x02	ChannelId field length is 4 bytes.
0x03	Invalid value.

ChannelId (variable): A variable length 8-bit, 16-bit, or 32-bit unsigned integer. Set to the value of the **ChannelId** associated with the DVC that is being closed.

3 Protocol Details

The following sections specify details of the Remote Desktop Protocol: Dynamic Virtual Channel Extension, including common, client, and server abstract data models and message processing rules.

3.1 Common Details

The following diagram illustrates the states and transitions of this protocol.

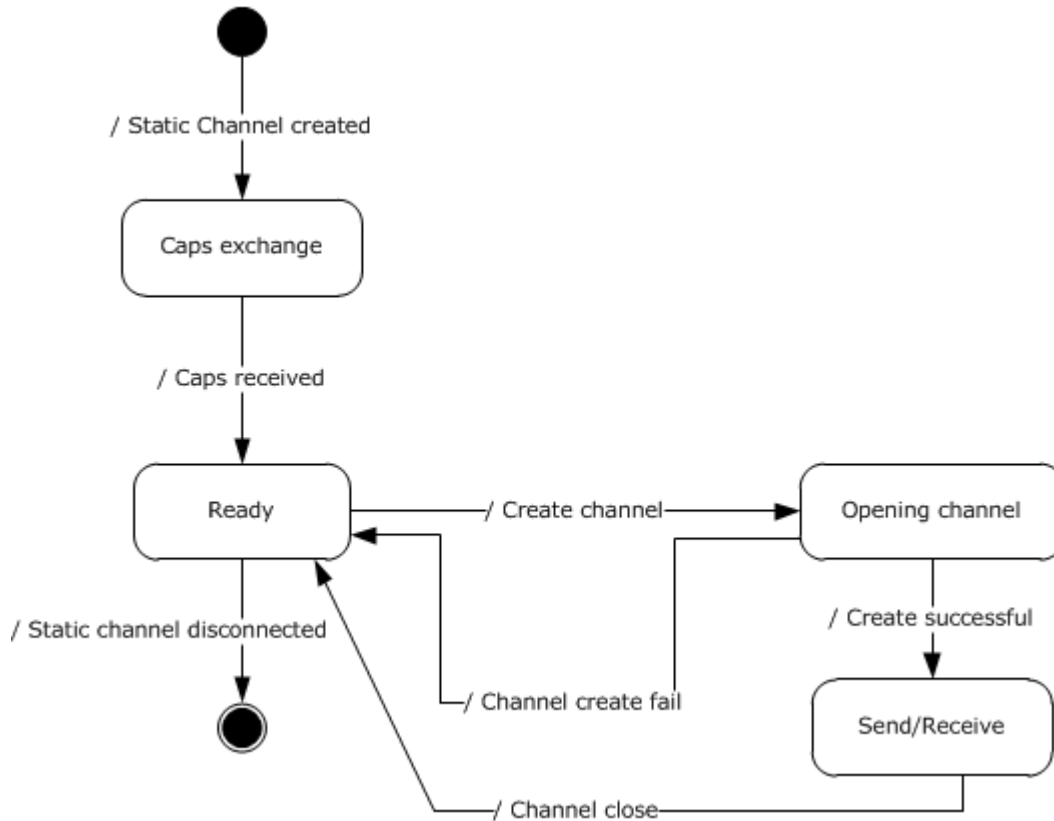


Figure 7: State diagram

Static channel created event: This event signals that the static channel is available to send and receive data, as described in [\[MS-RDPBCGR\]](#) section 3.1.5.2.

Caps exchange state: In this state, the client and the server exchange capabilities, as described in section [2.2.1](#).

Caps received event: After the capabilities have been received by both sides, the protocol is ready to open dynamic channels.

Ready state: In this state, the protocol can open new channels, as described in [2.2.2](#). For each new channel it will enter new state.

Create channel event: Signifies that the server is requesting a new channel, as described in section [2.2.2](#).

Opening channel state: In this state, the server awaits a DVC Create Response PDU, as described in section [2.2.2.2](#).

Channel create fail event: The response indicated failure, the channel is closed.

Create success event: The channel is ready to send and receive data.

Send/Receive state: The dynamic channel is ready to send and process incoming data. See below for detailed data processing state diagram.

Channel close event: Close the channel because either the client or the server have sent Closing a DVC packet, as described in section [2.2.4](#).

Static channel disconnected event: This event signifies that the connection between the client and the server is broken and, at this point, the protocol terminates.

The following diagram illustrates data processing while a channel is in the Send/Receive state.

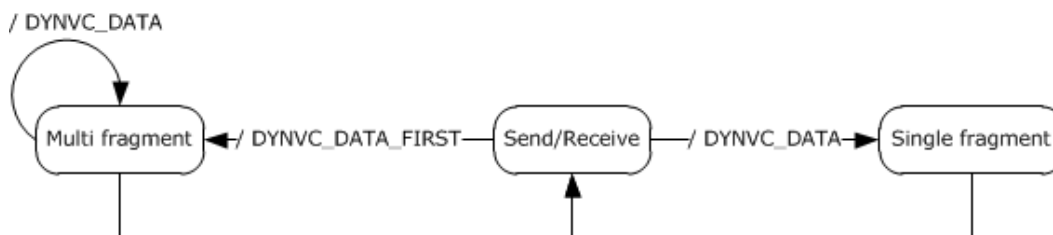


Figure 8: Data processing while a channel is in Send/Receive state

There are basically two substates:

Multi fragment state: This state starts with DYNVC_DATA_FIRST packet and then multiple DYNVC_DATA packets until the amount of data received is equal to the **Length** field of this packet.

Single fragment: Only a single DYNVC_DATA packet is received.

These reassembly rules are described in more detail in section [3.1.5.2.3](#).

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The organization is provided to explain how the protocol behaves. This document does not mandate that implementations adhere to this model, as long as their external behavior is consistent with that described in this document.

ChannelId: A unique ID that the server generates and sends in a [DVC Create Request PDU \(section 2.2.2.1\)](#). The **ChannelId** is valid until the server or client issues a Closing a DVC PDU (as specified in section [2.2.4](#)); afterwards, the **ChannelId** value can be reused in another DVC Create Request PDU.

Listener: A logical object identified by its name which consists of a string of ANSI encoded characters. The server uses the listener name to initiate channel connection by using a DVC Create Request PDU. The client maintains a list of active listeners and uses that list to send a [DVC Create Response PDU \(section 2.2.2.2\)](#). The client MAY change the list at any time without notifying the server. Changing the list content MUST NOT result in any change in the existing DVC state.

Priority Class: This field identifies how much bandwidth needs to be allocated for a particular class of channels. The bandwidth used by all channels with the same **Priority Class** MUST have a predetermined allocation, as described in section [2.2.1.1.2](#). **Priority Class** of value 0 will use the bandwidth calculated from PriorityCharge0, Priority Class of 1 will use PriorityCharge2, and so on.

Version: The number exchanged between clients in DVC Capabilities Request/Response packets, see section [2.2.1.1](#). It is used by both sides to determine what features the protocol supports as described in section [3.2.3.1](#).

3.1.2 Timers

No common timers are used.

3.1.3 Initialization

The TS server DVC manager begins the initialization sequence immediately following the establishment of a static virtual channel session, as specified in [\[MS-RDPBCGR\]](#) section 3.1.5.2. A TS server DVC manager and a TS client DVC manager exchange Capabilities PDUs as described in section [2.2.1](#) and initialize themselves to the version level negotiated. Individual DVCs are created by the DVC server manager exchanging the DVC Create Request/Response, as described in section [2.2.2](#), with the DVC client manager only after the Capabilities PDUs have been exchanged.

3.1.4 Higher-Layer Triggered Events

There are no common events specified for the Remote Desktop Protocol: Dynamic Virtual Channel Extension.

3.1.5 Message Processing Events and Sequencing Rules

Sending and receiving messages is symmetrical between the DVC server manager and the DVC client manager. After the server creates a DVC, applications running on either the server or the client can initiate sending a message. The PDUs and the sending sequence are the same regardless of who initiates sending the message.

Messages are sent and received using two different PDUs depending on the total length of the message being sent.

- [DYNVC DATA FIRST \(section 2.2.3.1\)](#)
- [DYNVC DATA \(section 2.2.3.2\)](#)

The maximum message size that an application can pass to a DVC manager for sending to a receiver at one time is $2^{32}-1$ bytes. The Remote Desktop Protocol: Dynamic Virtual Channel Extension specifies a maximum PDU size of 1,600 bytes. A data PDU consists of a header and message data. Large messages are fragmented by the sending DVC manager and reassembled by the receiving DVC manager. This is provided as a service to the requesting application.

Data transmitted over an unreliable channel MUST NOT be fragmented because an unreliable channel cannot guarantee that there will be no dropped or out-of-order sequence packets. In terms of the transports described in section [2.1](#), static virtual channels and UDP-R multitransport connections provide reliable data transportation, while UDP-L multitransport connections are unreliable and MUST NOT transport fragmented data.

3.1.5.1 Sending Data

3.1.5.1.1 DVC Data First (DYNVC_DATA_FIRST)

When the length of the message data being sent exceeds 1,590 bytes, the [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#) PDU is sent as the first data PDU. The **Length** field is set to the total length of the message, and the **Data** field contains the first block of fragmented data. Subsequent [DYNVC_DATA \(section 2.2.3.2\)](#) PDUs are sent until all the data has been sent.

3.1.5.1.2 DVC Data (DYNVC_DATA)

The [DYNVC_DATA \(section 2.2.3.2\)](#) PDU is used to send data when the total length of the PDU message plus the PDU header exceeds 1,600 bytes or the length of the message data being sent is less than 1,590 bytes. In the case where the total length of the PDU message plus the PDU header exceeds 1,600 bytes, the [DYNVC_DATA_FIRST \(2.2.3.1\)](#) PDU is sent as the first data PDU, followed by [DYNVC_DATA \(2.2.3.2\)](#) PDUs until all the data has been sent.

3.1.5.2 Receiving Data

Data transmitted over an unreliable channel MUST NOT be fragmented, because an unreliable channel cannot guarantee that there will be no dropped or out-of-order sequence packets. In terms of the transports described in [section 2.1](#), static virtual channels and UDP-R multitransport connections provide reliable data transportation, while UDP-L multitransport connections are unreliable and MUST NOT transport fragmented data.

3.1.5.2.1 DVC Data First (DYNVC_DATA_FIRST)

When the DVC manager receives a PDU, it checks the **Cmd** field to determine the type of PDU that has been sent. If the **Cmd** field is set to 0x02, the PDU type is [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#). The **Length** field indicates the total length of the message that is being sent, and the data field contains the first block of the fragmented message. The DVC manager stores the data and reads the next data PDU.

3.1.5.2.2 DVC Data (DYNVC_DATA)

When the DVC manager receives a PDU, it checks the **Cmd** field to determine the type of PDU that has been sent. If the **Cmd** field is set to 0x03, the PDU type is [DYNVC_DATA \(section 2.2.3.2\)](#). This PDU is used to send blocks of fragmented messages or one complete nonfragmented message.

If a [DYNVC_DATA \(section 2.2.3.2\)](#) PDU is preceded by a [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#) PDU, the receiver stores this data with the previously received data and continues to receive [DYNVC_DATA \(section 2.2.3.2\)](#) PDUs until the entire message has been received. The DVC manager reassembles the fragmented data and passes a complete message to the receiver.

If a [DYNVC_DATA \(section 2.2.3.2\)](#) PDU is not preceded by a [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#) PDU, the receiver passes the message in the **Data** field directly to the receiver without further processing.

3.1.5.2.3 Reassembly of Fragmented Virtual Channel Data

The DVC message can span multiple DVC data packets. If a transmission sequence starts with [DYNVC_DATA_FIRST \(section 2.2.3.1\)](#), this indicates a multiple-data packet transmission, with the total length of the data indicated by the Length field of [DYNVC_DATA_FIRST](#) packet.

The data arrives in multiple consecutive DYNVC_DATA (section [2.2.3.2](#)) packets until the total length of the message is received. At this point the message can be processed.

If a transmission sequence starts with DYNVC_DATA packet, then the DVC data message does not span multiple DVC packets and the data can be processed immediately.

3.1.5.2.4 Processing Packet Errors

When an unrecognized or malformed packet is received by the server or the client, the implementation **MUST** terminate the static virtual channel connection. This action **MUST** be taken for any unrecognized message or field values.

If a packet is received with a length shorter than the expected length, the implementation **MUST** terminate the static virtual channel connection.

Out-of-sequence packets, or the receipt of multiples of packets that are supposed to be received only once, **MUST** terminate the static virtual channel connection.

3.1.6 Timer Events

There are no common timer events.

3.1.7 Other Local Events

There are no local events specified for the Remote Desktop Protocol: Dynamic Virtual Channel Extension.

3.2 Client Details

3.2.1 Abstract Data Model

The abstract data model is specified in section [3.1.1](#).

3.2.2 Timers

There are no client timers.

3.2.3 Initialization

3.2.3.1 DVC Client Manager Initialization

The Remote Desktop Protocol: Dynamic Virtual Channel Extension encompasses two version levels. The server specifies priority charges in Version 2 (as described in section [2.2.1.1.2](#)); no such priority charges are specified in Version 1. If the client supports channel priorities it **MUST** set the **Version** field of [DYNVC_CAPS_RSP \(section 2.2.1.2\)](#) to 2.

Capabilities PDUs are exchanged to negotiate the version level of the Remote Desktop Protocol: Dynamic Virtual Channel Extension that is supported. Three different Capabilities PDUs are used to negotiate version level support.

- [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#) PDU is sent by a server to indicate it supports version 1 of this protocol.
- [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#) PDU is sent by a server to indicate it supports version 2 of this protocol.

- [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU is sent by a server to indicate it supports version 3 of this protocol.
- DYNVC_CAPS_RSP (section 2.2.1.2) PDU is sent by a client to acknowledge the version level it supports.

A Capabilities PDU has the **Cmd** field set to 0x05.

When a DVC client manager receives a Capabilities Request, it checks the **Version** field to determine what protocol version level the DVC server manager is requesting support for. The client **MUST** respond with a DYNVC_CAPS_RSP (section 2.2.1.2) PDU indicating the highest version level supported by the client.

3.2.3.1.1 Version Level 1 (DYNVC_CAPS_VERSION1)

A [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#) PDU has the **Version** field set to 0x01.

3.2.3.1.2 Version Level 2 (DYNVC_CAPS_VERSION2)

A [DYNVC_CAPS_VERSION2](#) PDU has the **Version** field set to 0x02.

A DYNVC_CAPS_VERSION2 (section 2.2.1.1.2) PDU specifies a message priority class. The client **SHOULD** use this information to allocate bandwidth for each priority class for the data that it sends to the server. Section [2.2.1.1.2](#) specifies how bandwidth is allocated based on the priority class.

3.2.3.1.3 Version Level 3 (DYNVC_CAPS_VERSION3)

A DYNVC_CAPS_VERSION3 PDU has the **Version** field set to 0x03.

A DYNVC_CAPS_VERSION3 (section [2.2.1.1.3](#)) PDU specifies a message priority class. The client **SHOULD** use this information to allocate bandwidth for each priority class for the data that it sends to the server. Section [2.2.1.1.2](#) specifies how bandwidth is allocated based on the priority class.

3.2.3.1.4 Capabilities Response (DYNVC_CAPS_RSP)

If a client receives a [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#), a [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#), or a [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU from the server, it **MUST** reply with a [Capabilities Response PDU \(section 2.2.1.2\)](#).

3.2.3.2 DVC Initialization

3.2.3.2.1 DVC Create Response (DYNVC_CREATE_RSP)

When a DVC client manager receives a [DYNVC_CREATE_REQ \(section 2.2.2.1\)](#) PDU from the DVC server manager, it uses the listener name and **ChannelId** to create a named channel. The DVC client manager uses the listener name to locate a listener on the TS client that has advertised itself as being available to accept connections. A client-side listener is created that associates the **ChannelId** with the specified listener. After the channel is created, the listener name is no longer used and all data is sent referencing the **ChannelId**.

The client maintains this channel for the life of the connection.

The client responds to the server with a [DYNVC_CREATE_RSP \(section 2.2.2.2\)](#) PDU indicating the channel creation status. Any positive or zero value indicates success. A negative value indicates failure.

3.2.4 Higher-Layer Triggered Events

Channel Close Request: Sent by a listener to the DVC client manager requesting that a DVC be closed.

Send-Data Request: A message passed to the DVC client manager for sending data over a DVC.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Sending and Receiving Data

Sending and receiving messages is symmetrical between the DVC server manager and the DVC client manager. After the server creates a DVC, applications running on either the server or the client can initiate sending a message. The PDUs and the sending sequence are the same regardless of who initiates sending the message. The sending and receiving of data is specified in section [3.1.5](#).

3.2.5.2 Closing a DVC (DYNVC_CLOSE)

A channel can be closed by either a DVC client manager or a DVC server manager. A [DYNVC_CLOSE \(section 2.2.4\)](#) PDU is used for both a close request and a close response.

When a DVC client manager receives a DYNVC_CLOSE (section 2.2.4) PDU, the client MAY respond with a DYNVC_CLOSE (section 2.2.4) PDU specifying the **ChannelId**.

When a client initiates a channel-close, it sends an unsolicited DYNVC_CLOSE (section 2.2.4) PDU specifying the **ChannelId** to the server. The server does not respond to the DYNVC_CLOSE (section 2.2.4) PDU.

Upon closing a channel, the client SHOULD remove the **ChannelId** from the list of active **ChannelIds**.

3.2.6 Timer Events

There are no client timer events.

3.2.7 Other Local Events

There are no other local events used by the client.

3.3 Server Details

3.3.1 Abstract Data Model

The abstract data model is specified in section [3.1.1](#).

3.3.2 Timers

The DVC server manager sets a timer when it is initializing a DVC. The timer is set to expire 10 seconds after the server sends a Capabilities Request to the DVC client manager. The server sends either a [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#), a [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#), or a [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU, depending on the protocol version level it supports. If the client does not respond to the Capabilities Request within 10 seconds, the server fails the creation of the DVC.

3.3.3 Initialization

Before protocol operation can commence, the static virtual channel MUST be established using the parameters specified in section [2.1](#).

3.3.3.1 DVC Server Manager Initialization

The DVC managers on the TS server and the TS client initialize themselves by exchanging Capabilities PDUs to negotiate the version level of the Remote Desktop Protocol: Dynamic Virtual Channel Extension that is supported. Three different Capabilities PDUs are used to negotiate version level support:

- [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#) PDU is sent by a server to indicate it supports version 1 of this protocol.
- [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#) PDU is sent by a server to indicate it supports version 2 of this protocol.
- [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU is sent by a server to indicate it supports version 3 of this protocol.
- [DYNVC_CAPS_RSP \(section 2.2.1.2\)](#) PDU is sent by a client to acknowledge the version level it supports.

A Capabilities PDU has the **Cmd** field set to 0x05.

Immediately following the establishment of a static virtual channel session, as specified in [\[MS-RDPBCGR\]](#), the DVC server manager begins the initialization sequence. The DVC server manager sets a 10-second timer and sends a [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#), a [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#), or a [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU to the DVC client manager to indicate its highest version level supported. The DVC client manager MUST respond with a [DYNVC_CAPS_RSP \(section 2.2.1.2\)](#) PDU indicating the highest version level supported by the client. If the client does not respond to a Capabilities Request PDU before the 10-second timer expires, the server fails the creation of the DVC.

The DVC server manager MUST adjust to the version level supported by the DVC client manager.

3.3.3.1.1 Version Level 1 (DYNVC_CAPS_VERSION1)

A [DYNVC_CAPS_VERSION1 \(section 2.2.1.1.1\)](#) PDU is sent to indicate that the highest protocol version level that the DVC server manager supports is version level 1. For PDU initialization see section [Version Level 1 \(DYNVC_CAPS_VERSION1\) \(section 3.2.3.1.1\)](#).

3.3.3.1.2 Version Level 2 (DYNVC_CAPS_VERSION2)

A [DYNVC_CAPS_VERSION2 \(section 2.2.1.1.2\)](#) PDU is sent to indicate that the highest protocol version level the DVC server manager supports is version level 2. For PDU initialization see section [Version Level 2 \(DYNVC_CAPS_VERSION2\) \(section 3.2.3.1.2\)](#).

3.3.3.1.3 Version Level 3 (DYNVC_CAPS_VERSION3)

A [DYNVC_CAPS_VERSION3 \(section 2.2.1.1.3\)](#) PDU is sent to indicate that the highest protocol version level the DVC server manager supports is version level 3. For PDU initialization, see [Version Level 3 \(DYNVC_CAPS_VERSION3\) \(section 3.2.3.1.3\)](#).

3.3.3.1.4 Capabilities Response (DYNVC_CAPS_RSP)

The [DYNVC_CAPS_RSP \(section 2.2.1.2\)](#) PDU is sent by the DVC client manager to indicate the protocol version level it supports.

The version level supported is indicated by the value of the **Version** field. If the client doesn't respond with a Capabilities Response packet within 10 seconds, the server MUST NOT make any further attempts to send a DVC Create Request packet.

3.3.3.2 DVC Initialization

A DVC server manager initializes a channel by exchanging Create PDUs with a DVC client manager.

Two different Create PDUs are used to open a channel:

- [DYNVC_CREATE_REQ \(section 2.2.2.1\)](#)
- [DYNVC_CREATE_RSP \(section 2.2.2.2\)](#)

A DVC server manager creates a channel in response to a request by an application running on the TS server to create a channel to a listener running on the TS client. The server creates a channel for the requesting application using a listener name. The server sends the **ChannelId** and the listener name to the DVC client manager. A version 2 server also sends the channel priority to the DVC client manager. The server maintains this **ChannelId** for the life of the channel.

The DVC client manager responds to the DVC server manager with a DYNVC_CREATE_RSP (section 2.2.2.2) PDU indicating the channel creation status. Any positive or zero value indicates success. A negative value indicates failure.

The listener name is only used at channel initialization. After the channel is initialized, all data is sent referencing the **ChannelId**.

The Channel ID MUST be unique within a static virtual channel session.

There is no processing specified by the Remote Desktop Protocol: Dynamic Virtual Channel Extension for a priority class.

3.3.4 Higher-Layer Triggered Events

- Static virtual channel created: Immediately upon creation of a static virtual channel, as specified in [\[MS-RDPBCGR\]](#), the DVC server manager begins the initialization sequence.
- Channel creation request: Sent by an application to the DVC server manager requesting the creation of a DVC with a listener running on a TS client.
- Channel close request: Sent by an application to the DVC server manager requesting the closure of a DVC.
- Send-Data request: A message passed to the DVC server manager for sending data over a DVC.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Sending and Receiving Data

Sending and receiving messages is symmetrical between the DVC server manager and the DVC client manager. After the server creates a DVC, applications running on either the server or the

client can initiate sending a message. The PDUs and the sending sequence are the same regardless of who initiates sending the message. The sending and receiving of data is specified in section [3.1.5](#).

3.3.5.2 Closing a DVC (DYNVC_CLOSE)

The closure of a channel can be requested by either an application running on the TS server or a listener running on the TS client. A [DYNVC_CLOSE \(section 2.2.4\)](#) PDU is used for both a close request and a close response.

The DVC server manager sends a DYNVC_CLOSE (section 2.2.4) PDU (specifying the **ChannelId** to close) to the DVC client manager. The client replies with a DYNVC_CLOSE (section 2.2.4) PDU.

When a DVC client manager initiates a channel-close, it sends an unsolicited DYNVC_CLOSE (section 2.2.4) PDU specifying the **ChannelId** to the server. The server does not respond to the DYNVC_CLOSE (section 2.2.4) PDU.

Upon closing the channel, the DVC server manager MAY reuse the **ChannelId** in the next [DVC Create Request PDU \(section 2.2.2.1\)](#).

3.3.6 Timer Events

There are no timer events used by the Remote Desktop Protocol: Dynamic Virtual Channel Extension.

3.3.7 Other Local Events

There are no other local events.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Remote Desktop Protocol: Dynamic Virtual Channel Extension Specification.

4.1 Annotated Initializing DVCs

4.1.1 DVC Capabilities Request (Version2) PDU

The following is an annotated sample of the [DYNVC_CAPS_VERSION2 PDU \(section 2.2.1.1.2\)](#).

```
00000000 58 00 02 00 33 33 11 11 3d 0a a7 04          X...33..=...

58 -> Header bitmask fields

0 - --\
1 -   | DYNVC_CAPS_VERSION2::Cmd = Capabilities (5)
0 -   |
1 - --/
1 - --\ DYNVC_CAPS_VERSION2::Sp = 2
0 - --/
0 - --\ DYNVC_CAPS_VERSION2::cbChId = 0
0 - --/

00 -> Pad

02 00 -> DYNVC_CAPS_VERSION2::Version = 0x0002 = 2

33 33 -> DYNVC_CAPS_VERSION2::PriorityCharge0 = 0x3333 = 13107 (~5%)
11 11 -> DYNVC_CAPS_VERSION2::PriorityCharge1 = 0x1111 = 4369 (~15%)
3d 0a -> DYNVC_CAPS_VERSION2::PriorityCharge2 = 0x0a3d = 2621 (~25%)
a7 04 -> DYNVC_CAPS_VERSION2::PriorityCharge3 = 0x04a7 = 1191 (~55%)
```

4.1.2 DVC Capabilities Response PDU

The following is an annotated sample of the [DYNVC_CAPS_RSP PDU \(section 2.2.1.2\)](#).

```
00000000 50 00 02 00

50 -> Header bitmask fields

0 - --\
1 -   | DYNVC_CAPS_RSP::Cmd = Capabilities (5)
0 -   |
1 - --/
0 - --\ DYNVC_CAPS_RSP::Sp = 0
0 - --/
0 - --\ DYNVC_CAPS_RSP::cbChId = 0
0 - --/

00 -> Pad

02 00 -> DYNVC_CAPS_RSP::Version = 0x0002 = 2
```



```
71 .... 71 -> DYNVC_DATA::Data = (1600 - 2) bytes of actual payload
```

The last packet in the sequence for the remaining data: DYNVC_DATA PDU (section 2.2.3.2).

```
00000000 34 03 71
```

```
34 -> Header bitmask fields
```

```
0 - --\  
0 - | DYNVC_DATA::Cmd = Data (3)  
1 - |  
1 - --/  
0 - --\  
0 - | DYNVC_DATA::Sp = 1  
1 - |  
1 - --/  
0 - --\  
0 - | DYNVC_DATA::cbChId = 0  
0 - |  
0 - --/
```

```
03 -> DYNVC_DATA::ChannelId = 0x3
```

```
71 -> DYNVC_DATA::Data = (3 - 2) bytes of actual payload.
```

4.4 Annotated Closing a DVC

4.4.1 DVC Close PDU

The following is an annotated sample of the [DYNVC_CLOSE PDU \(section 2.2.4\)](#).

```
00000000 40 03 @.
```

```
40 -> Header bitmask fields
```

```
0 - --\  
1 - | DYNVC_CLOSE::Cmd = Close (4)  
0 - |  
0 - --/  
0 - --\  
0 - | DYNVC_CLOSE::Sp = 0  
0 - |  
0 - --/  
0 - --\  
0 - | DYNVC_CLOSE::cbChId = 0  
0 - |  
0 - --/
```

```
03 -> DYNVC_CLOSE::ChannelId = 0x3
```

5 Security

The following sections specify security considerations for implementers of the Remote Desktop Protocol: Dynamic Virtual Channel Extension.

5.1 Security Considerations for Implementers

There are no security considerations for Remote Desktop Protocol: Dynamic Virtual Channel Extension messages because all static virtual channel traffic is secured by the underlying RDP core protocol. For an overview of the implemented security-related mechanisms, see [\[MS-RDPBCGR\]](#) section 5.

5.2 Index of Security Parameters

There are no security parameters in the Remote Desktop Protocol: Dynamic Virtual Channel Extension.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

[<1> Section 2.2.1.1.1:](#) The DYNVC_CAPS_VERSION1 PDU is supported only in Windows Vista.

[<2> Section 2.2.1.1.1:](#) Windows implementations initialize **Sp** to 0x02.

[<3> Section 2.2.1.1.2:](#) The [DYNVC_CAPS_VERSION2](#) PDU is supported in Windows Vista SP1, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

[<4> Section 2.2.1.1.2:](#) Windows implementations initialize Sp to 0x02.

[<5> Section 2.2.1.1.3:](#) The DYNVC_CAPS_VERSION3 PDU is supported in Windows 8, Windows Server 2012, Windows 8.1, and Windows Server 2012 R2.

[<6> Section 2.2.1.1.3:](#) Windows implementations initialize Sp to 0x02.

[<7> Section 2.2.3.2:](#) Windows implementations do not initialize this field. As a result, its value is random.

7 Change Tracking

This section identifies changes that were made to the [MS-RDPEDYC] protocol document between the November 2013 and February 2014 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.

- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.5.1.2 DVC Data (DYNVC_DATA)	69300 Updated behavior of the DYNVC_DATA PDU.	Y	Content updated.

8 Index

A

Abstract data model
 client ([section 3.1.1](#) 27, [section 3.2.1](#) 30)
 server ([section 3.1.1](#) 27, [section 3.3.1](#) 32)
[Annotated closing DVC](#) 42
[Annotated initializing DVC](#) 36
[Annotated opening DVC](#) 37
[Annotated sending and receiving data](#) 37
[Applicability](#) 13

C

[Capability negotiation](#) 13
[Change tracking](#) 45
Client ([section 3.1](#) 26, [section 3.2](#) 30)
 abstract data model ([section 3.1.1](#) 27, [section 3.2.1](#) 30)
 higher-layer triggered events ([section 3.1.4](#) 28, [section 3.2.4](#) 32)
 initialization ([section 3.1.3](#) 28, [section 3.2.3](#) 30)
 message processing ([section 3.1.5](#) 28, [section 3.2.5](#) 32)
 sequencing rules ([section 3.1.5](#) 28, [section 3.2.5](#) 32)
 timer events ([section 3.1.6](#) 30, [section 3.2.6](#) 32)
 timers ([section 3.1.2](#) 28, [section 3.2.2](#) 30)

D

Data
 receiving ([section 1.3.3.2](#) 11, [section 1.3.3.2.2](#) 12, [section 2.2.3](#) 22, [section 3.1.5.2](#) 29, [section 3.2.5.1](#) 32, [section 3.3.5.1](#) 34)
 sending ([section 1.3.3.2](#) 11, [section 1.3.3.2.1](#) 12, [section 2.2.3](#) 22, [section 3.1.5.1](#) 29, [section 3.2.5.1](#) 32, [section 3.3.5.1](#) 34)
Data model - abstract
 client ([section 3.1.1](#) 27, [section 3.2.1](#) 30)
 server ([section 3.1.1](#) 27, [section 3.3.1](#) 32)
[DRDYNVC Static Virtual Channel](#) 9
DVC
 [capabilities request](#) 17
 [capabilities response](#) 20
 [client manager initialization](#) 30
 closing ([section 1.3.3.3](#) 13, [section 2.2.4](#) 24)
 [create request](#) 20
 [create response](#) 21
 initialization ([section 3.2.3.2](#) 31, [section 3.3.3.2](#) 34)
 [initializing](#) 16
 [message flow](#) 10
 message flow - receiving data ([section 1.3.3.2](#) 11, [section 1.3.3.2.2](#) 12)
 message flow - sending data ([section 1.3.3.2](#) 11, [section 1.3.3.2.1](#) 12)
 [opening](#) 20
 [server manager initialization](#) 33

[setup](#) 9
 [DVC traffic](#) 9
 [DYNVC CAPS_RSP packet](#) 20
 [DYNVC CAPS_VERSION1 packet](#) 17
 [DYNVC CAPS_VERSION2 packet](#) 17
 [DYNVC CAPS_VERSION3 packet](#) 19
 [DYNVC_CLOSE packet](#) 24
 [DYNVC_CREATE_REQ packet](#) 20
 [DYNVC_CREATE_RSP packet](#) 21
 [DYNVC_DATA packet](#) 23
 [DYNVC_DATA_FIRST packet](#) 22

E

encapsulation
 [DRDYNVC Static Virtual Channel](#) 9
Examples
 [annotated closing DVC](#) 42
 [annotated initializing DVC](#) 36
 [annotated opening DVC](#) 37
 [annotated sending and receiving data](#) 37
 [overview](#) 36

F

[Fields - vendor-extensible](#) 14

G

[Glossary](#) 7

H

Higher-layer triggered events
 client ([section 3.1.4](#) 28, [section 3.2.4](#) 32)
 server ([section 3.1.4](#) 28, [section 3.3.4](#) 34)

I

[Implementer - security considerations](#) 43
[Index of security parameters](#) 43
[Informative references](#) 8
Initialization
 client ([section 3.1.3](#) 28, [section 3.2.3](#) 30)
 server ([section 3.1.3](#) 28, [section 3.3.3](#) 33)
[Introduction](#) 7

M

Message flow
 [DVC](#) 10
 DVC - receiving data ([section 1.3.3.2](#) 11, [section 1.3.3.2.2](#) 12)
 DVC - sending data ([section 1.3.3.2](#) 11, [section 1.3.3.2.1](#) 12)
Message processing
 client ([section 3.1.5](#) 28, [section 3.2.5](#) 32)
 server ([section 3.1.5](#) 28, [section 3.3.5](#) 34)

[Message Syntax packet](#) 15

Messages
[overview](#) 15
[syntax](#) 15
[transport](#) 15

N

[Normative references](#) 8

O

[Overview](#) 8

P

[Parameters - security index](#) 43
[Preconditions](#) 13
[Prerequisites](#) 13
[Product behavior](#) 44

R

References
[informative](#) 8
[normative](#) 8
[Relationship to other protocols](#) 13

S

Security
[implementer considerations](#) 43
[overview](#) 43
[parameter index](#) 43

Sequencing rules
client ([section 3.1.5](#) 28, [section 3.2.5](#) 32)
server ([section 3.1.5](#) 28, [section 3.3.5](#) 34)

Server ([section 3.1](#) 26, [section 3.3](#) 32)
abstract data model ([section 3.1.1](#) 27, [section 3.3.1](#) 32)
higher-layer triggered events ([section 3.1.4](#) 28, [section 3.3.4](#) 34)
initialization ([section 3.1.3](#) 28, [section 3.3.3](#) 33)
message processing ([section 3.1.5](#) 28, [section 3.3.5](#) 34)
sequencing rules ([section 3.1.5](#) 28, [section 3.3.5](#) 34)
timer events ([section 3.1.6](#) 30, [section 3.3.6](#) 35)
timers ([section 3.1.2](#) 28, [section 3.3.2](#) 32)

Set up
[DVC](#) 9
[Standards assignments](#) 14
[Syntax - message](#) 15

T

Timer events
client ([section 3.1.6](#) 30, [section 3.2.6](#) 32)
server ([section 3.1.6](#) 30, [section 3.3.6](#) 35)

Timers
client ([section 3.1.2](#) 28, [section 3.2.2](#) 30)
server ([section 3.1.2](#) 28, [section 3.3.2](#) 32)

[Tracking changes](#) 45

[Transport - message](#) 15
Triggered events - higher-layer
client ([section 3.1.4](#) 28, [section 3.2.4](#) 32)
server ([section 3.1.4](#) 28, [section 3.3.4](#) 34)

V

[Vendor-extensible fields](#) 14
[Versioning](#) 13