[MS-RDPEI]: Remote Desktop Protocol: Input 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
12/16/2011	1.0	New	Released new document.
03/30/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	2.0	Major	Significantly changed the technical content.
10/25/2012	3.0	Major	Significantly changed the technical content.
01/31/2013	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	3.1	Minor	Clarified the meaning of the technical content.

Contents

[MS-RDPEI] — v20130722
Remote Desktop Protocol: Input Virtual Channel Extension

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

Copyright © 2013 Microsoft Corporation.

1 Introduction

The Remote Desktop Protocol: Input Virtual Channel Extension applies to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, as specified in [MS-RDPBCGR] sections 1 to 5. The input protocol defined in section 2.2 is used to remote multitouch input frames from a **terminal server** client to a terminal server. The multitouch input frames are generated at the client by a physical or virtual touch digitizer, encoded, and then sent on the wire to the server. After this input is received and decoded by the server, it is injected into the session associated with the remote user, effectively remoting the multitouch input frames generated at the client.

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]:

little-endian
PDU

The following terms are specific to this document:

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

terminal server: The server to which the client initiated the remote desktop connection.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described 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. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

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

[MS-RDPEDYC] Microsoft Corporation, "Remote Desktop Protocol: Dynamic Channel Virtual Channel Extension".

[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

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

1.3 Overview

An example message flow encapsulating all of the Input Messages described in section 2.2.3 and protocol phases is presented in the following figure.

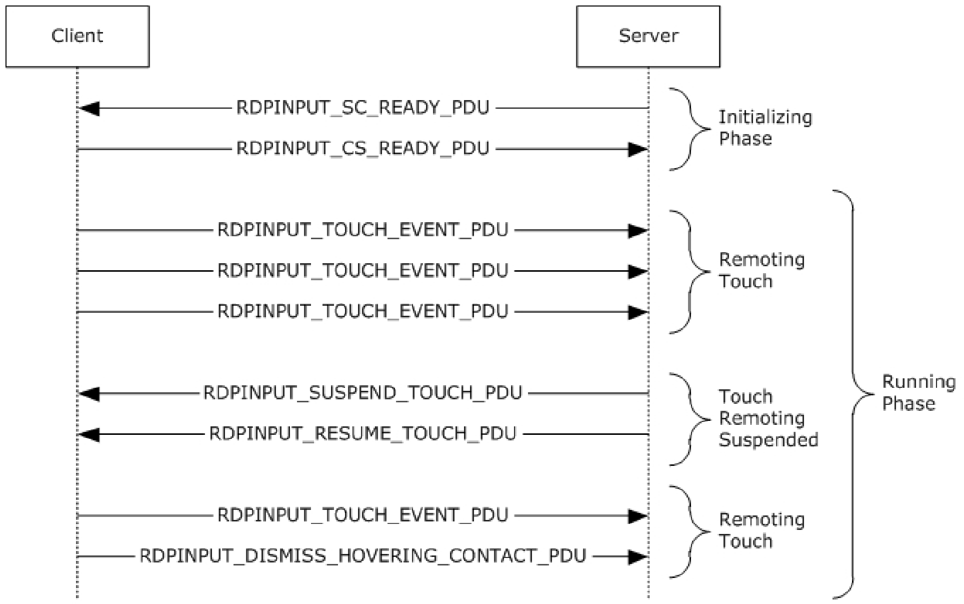


Figure 1: Messages exchanged by the Input Protocol endpoints

The Input Protocol is divided into two distinct phases:

- Initializing Phase
- Running Phase

The Initializing Phase occurs at the start of the connection. During this phase, the server and client exchange the RDPINPUT_SC_READY_PDU (section 2.2.3.1) and RDPINPUT_CS_READY_PDU (section 2.2.3.2) messages. The server initiates this exchange when the dynamic virtual channel (sections 1.4 and 2.1) over which the Input Messages will flow has been opened.

Once both endpoints are ready, the Running Phase is entered. During this phase, the client sends touch frames to the server encapsulated in the RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.2) message. The server decodes these touch frames and injects them into the user's session.

During the Running Phase, the server can request that the client suspend the transmission of touch frames by sending the RDPINPUT_SUSPEND_TOUCH_PDU (section 2.2.3.3) message to the client.

Copyright © 2013 Microsoft Corporation.

To request that the client resume the transmission of touch frames, the server can send the RDPINPUT_RESUME_TOUCH_PDU (section 2.2.3.4) message to the client.

To transition contacts in the "hovering" state to the "out of range" state (section 3.1.1.1), the client can send the RDPINPUT_DISMISS_HOVERING_CONTACT_PDU (section 2.2.3.6) message to the server. This message effectively allows individual contacts (in the hovering state) to be transitioned to the out of range state without requiring the construction and transmission of a touch frame from client to server. If the contact specified in the RDPINPUT_DISMISS_HOVERING_CONTACT_PDU message does not exist on the server, then the message is simply ignored.

1.4 Relationship to Other Protocols

The Remote Desktop Protocol: Input Virtual Channel Extension is embedded in a dynamic virtual channel transport, as specified in [MS-RDPEDYC] sections 1 to 3.

1.5 Prerequisites/Preconditions

The Remote Desktop Protocol: Input Virtual Channel Extension operates only after the dynamic virtual channel transport is fully established. If the dynamic virtual channel transport is terminated, the Remote Desktop Protocol: Input Virtual Channel Extension is also terminated. The protocol is terminated by closing the underlying virtual channel. For details about closing the dynamic virtual channel, see [MS-RDPEDYC] section 3.2.5.2.

1.6 Applicability Statement

The Remote Desktop Protocol: Input Virtual Channel Extension is applicable in scenarios where the transfer of multitouch input frames (generated by a physical or virtual touch digitizer) is required from a terminal server client to a terminal server.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

Copyright © 2013 Microsoft Corporation.

2 Messages

2.1 Transport

The Remote Desktop Protocol: Input Virtual Channel Extension is designed to operate over a dynamic virtual channel, as specified in [MS-RDPEDYC] sections 1 to 3. The dynamic virtual channel name is the null-terminated **ANSI character** string "Microsoft::Windows::RDS::Input". The usage of channel names in the context of opening a dynamic virtual channel is specified in [MS-RDPEDYC] section 2.2.2.1.

2.2 Message Syntax

The following sections specify the Remote Desktop Protocol: Input Virtual Channel Extension message syntax. All multiple-byte fields within a message MUST be marshaled in **little-endian** byte order, unless otherwise specified.

2.2.1 Namespaces

2.2.2 Common Data Types

2.2.2.1 TWO_BYTE_UNSIGNED_INTEGER

The **TWO_BYTE_UNSIGNED_INTEGER** structure is used to encode a value in the range 0x0000 to 0x7FFF by using a variable number of bytes. For example, 0x1A1B is encoded as { 0x9A, 0x1B }. The most significant bit of the first byte encodes the number of bytes in the structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
c	val1							val2 (optional)																							

c (1 bit): A 1-bit unsigned integer field containing an encoded representation of the number of bytes in this structure.

Value	Meaning
ONE_BYTE_VAL 0	Implies that the optional **val2** field is not present. Hence, the structure is 1 byte in size.
TWO_BYTE_VAL 1	Implies that the optional **val2** field is present. Hence, the structure is 2 bytes in size.

val1 (7 bits): A 7-bit unsigned integer field containing the most significant 7 bits of the value represented by this structure.

val2 (1 byte, optional): An 8-bit unsigned integer containing the least significant bits of the value represented by this structure.

2.2.2.2 TWO_BYTE_SIGNED_INTEGER

The **TWO_BYTE_SIGNED_INTEGER** structure is used to encode a value in the range -0x3FFF to 0x3FFF by using a variable number of bytes. For example, -0x1A1B is encoded as { 0xDA, 0x1B },

Copyright © 2013 Microsoft Corporation.

and -0x0002 is encoded as { 0x42 }. The most significant bits of the first byte encode the number of bytes in the structure and the sign.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
c	s	val1						val2 (optional)																							

c (1 bit): A 1-bit unsigned integer field containing an encoded representation of the number of bytes in this structure.

Value	Meaning
ONE_BYTE_VAL 0	Implies that the optional **val2** field is not present. Hence, the structure is 1 byte in size.
TWO_BYTE_VAL 1	Implies that the optional **val2** field is present. Hence, the structure is 2 bytes in size.

s (1 bit): A 1-bit unsigned integer field containing an encoded representation of whether the value is positive or negative.

Value	Meaning
POSITIVE_VAL 0	Implies that the value represented by this structure is positive.
NEGATIVE_VAL 1	Implies that the value represented by this structure is negative.

val1 (6 bits): A 6-bit unsigned integer field containing the most significant 6 bits of the value represented by this structure.

val2 (1 byte, optional): An 8-bit unsigned integer containing the least significant bits of the value represented by this structure.

2.2.2.3 FOUR_BYTE_UNSIGNED_INTEGER

The **FOUR_BYTE_UNSIGNED_INTEGER** structure is used to encode a value in the range 0x00000000 to 0x3FFFFFFF by using a variable number of bytes. For example, 0x001A1B1C is encoded as {0x9A, 0x1B, 0x1C}. The two most significant bits of the first byte encode the number of bytes in the structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
c		val1						val2 (optional)								val3 (optional)								val4 (optional)							

c (2 bits): A 2-bit unsigned integer field containing an encoded representation of the number of bytes in this structure.

Value	Meaning
ONE_BYTE_VAL 0	Implies that the optional **val2**, **val3**, and **val4** fields are not present. Hence, the structure is 1 byte in size.
TWO_BYTE_VAL 1	Implies that the optional **val2** field is present, while the optional **val3** and **val4** fields are not present. Hence, the structure is 2 bytes in size.
THREE_BYTE_VAL 2	Implies that the optional **val2** and **val3** fields are present, while the optional **val4** field is not present. Hence, the structure is 3 bytes in size.
FOUR_BYTE_VAL 3	Implies that the optional **val2**, **val3**, and **val4** fields are all present. Hence, the structure is 4 bytes in size.

val1 (6 bits): A 6-bit unsigned integer field containing the most significant 6 bits of the value represented by this structure.

val2 (1 byte, optional): An 8-bit unsigned integer containing the second most significant bits of the value represented by this structure.

val3 (1 byte, optional): An 8-bit unsigned integer containing the third most significant bits of the value represented by this structure.

val4 (1 byte, optional): An 8-bit unsigned integer containing the least significant bits of the value represented by this structure.

2.2.2.4 FOUR_BYTE_SIGNED_INTEGER

The **FOUR_BYTE_SIGNED_INTEGER** structure is used to encode a value in the range -0x1FFFFFFF to 0x1FFFFFFF by using a variable number of bytes. For example, -0x001A1B1C is encoded as {0xBA, 0x1B, 0x1C}, and -0x00000002 is encoded as {0x22}. The three most significant bits of the first byte encode the number of bytes in the structure and the sign.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
c		s	val1					val2 (optional)								val3 (optional)								val4 (optional)							

c (2 bits): A 2-bit unsigned integer field containing an encoded representation of the number of bytes in this structure.

Value	Meaning
ONE_BYTE_VAL 0	Implies that the optional **val2**, **val3**, and **val4** fields are not present. Hence, the structure is 1 byte in size.
TWO_BYTE_VAL 1	Implies that the optional **val2** field is present, while the optional **val3** and **val4** fields are not present. Hence, the structure is 2 bytes in size.
THREE_BYTE_VAL 2	Implies that the optional **val2** and **val3** fields are present, while the optional **val4** field is not present. Hence, the structure is 3 bytes in size.
FOUR_BYTE_VAL	Implies that the optional **val2**, **val3**, and **val4** fields are all present. Hence,

Copyright © 2013 Microsoft Corporation.

Value	Meaning
3	the structure is 4 bytes in size.

s (1 bit): A 1-bit unsigned integer field containing an encoded representation of whether the value is positive or negative.

Value	Meaning
POSITIVE_VAL 0	Implies that the value represented by this structure is positive.
NEGATIVE_VAL 1	Implies that the value represented by this structure is negative.

val1 (5 bits): A 5-bit unsigned integer field containing the most significant 5 bits of the value represented by this structure.

val2 (1 byte, optional): An 8-bit unsigned integer containing the second most significant bits of the value represented by this structure.

val3 (1 byte, optional): An 8-bit unsigned integer containing the third most significant bits of the value represented by this structure.

val4 (1 byte, optional): An 8-bit unsigned integer containing the least significant bits of the value represented by this structure.

2.2.2.5 EIGHT_BYTE_UNSIGNED_INTEGER

The EIGHT_BYTE_UNSIGNED_INTEGER structure is used to encode a value in the range 0x0000000000000000 to 0x1FFFFFFFFFFFFFFF by using a variable number of bytes. For example, 0x001A1B1C1D1E1F2A is encoded as {0xDA, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x2A}. The three most significant bits of the first byte encode the number of bytes in the structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
c			val1					val2 (optional)								val3 (optional)								val4 (optional)							
val5 (optional)								val6 (optional)								val7 (optional)								val8 (optional)							

c (3 bits): A 3-bit unsigned integer field containing an encoded representation of the number of bytes in this structure.

Value	Meaning
ONE_BYTE_VAL 0	Implies that the optional **val2**, **val3**, **val4, val5**, **val6**, **val7** and **val8** fields are not present. Hence, the structure is 1 byte in size.
TWO_BYTE_VAL 1	Implies that the optional **val2** field is present, while the optional **val3**, **val4, val5**, **val6**, **val7** and **val8** fields are not present. Hence, the structure is 2 bytes in size.

Value	Meaning
THREE_BYTE_VAL 2	Implies that the optional **val2** and **val3** fields are present, while the optional **val4, val5**, **val6**, **val7** and **val8** fields are not present. Hence, the structure is 3 bytes in size.
FOUR_BYTE_VAL 3	Implies that the optional **val2**, **val3**, and **val4** fields are all present, while the optional **val5**, **val6**, **val7** and **val8** fields are not present. Hence, the structure is 4 bytes in size.
FIVE_BYTE_VAL 4	Implies that the optional **val2**, **val3**, **val4** and **val5** fields are all present, while the optional **val6**, **val7** and **val8** fields are not present. Hence, the structure is 5 bytes in size.
SIX_BYTE_VAL 5	Implies that the optional **val2**, **val3**, **val4**, **val5** and **val6** fields are all present, while the optional **val7** and **val8** fields are not present. Hence, the structure is 6 bytes in size.
SEVEN_BYTE_VAL 6	Implies that the optional **val2**, **val3**, **val4**, **val5**, **val6** and **val7** fields are all present, while the optional **val8** field is not present. Hence, the structure is 7 bytes in size.
EIGHT_BYTE_VAL 7	Implies that the optional **val2**, **val3**, **val4**, **val5**, **val6**, **val7** and **val8** fields are all present. Hence, the structure is 8 bytes in size.

val1 (5 bits): A 5-bit unsigned integer field containing the most significant 5 bits of the value represented by this structure.

val2 (1 byte, optional): An 8-bit unsigned integer containing the second most significant bits of the value represented by this structure.

val3 (1 byte, optional): An 8-bit unsigned integer containing the third most significant bits of the value represented by this structure.

val4 (1 byte, optional): An 8-bit unsigned integer containing the fourth most significant bits of the value represented by this structure.

val5 (1 byte, optional): An 8-bit unsigned integer containing the fifth most significant bits of the value represented by this structure.

val6 (1 byte, optional): An 8-bit unsigned integer containing the sixth most significant bits of the value represented by this structure.

val7 (1 byte, optional): An 8-bit unsigned integer containing the seventh most significant bits of the value represented by this structure.

val8 (1 byte, optional): An 8-bit unsigned integer containing the least significant bits of the value represented by this structure.

2.2.2.6 RDPINPUT_HEADER

The **RDPINPUT_HEADER** structure is included in all input event **PDU**s and is used to identify the input event type and to specify the length of the PDU.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
eventId																pduLength															
...																															

eventId (2 bytes): A 16-bit unsigned integer that identifies the type of the input event PDU.

Value	Meaning
EVENTID_SC_READY 0x0001	RDPINPUT_SC_READY_PDU (section 2.2.3.1).
EVENTID_CS_READY 0x0002	RDPINPUT_CS_READY_PDU (section 2.2.3.2)
EVENTID_TOUCH 0x0003	RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.3)
EVENTID_SUSPEND_TOUCH 0x0004	RDPINPUT_SUSPEND_TOUCH_PDU (section 2.2.3.4).
EVENTID_RESUME_TOUCH 0x0005	RDPINPUT_RESUME_TOUCH_PDU (section 2.2.3.5).
EVENTID_DISMISS_HOVERING_CONTACT 0x0006	RDPINPUT_DISMISS_HOVERING_CONTACT_PDU (section 2.2.3.6)

pduLength (4 bytes): A 32-bit unsigned integer that specifies the length of the input event PDU in bytes. This value MUST include the length of the **RDPINPUT_HEADER** (6 bytes).

2.2.3 Input Messages

2.2.3.1 RDPINPUT_SC_READY_PDU

The **RDPINPUT_SC_READY_PDU** message is sent by the server endpoint and is used to indicate readiness to commence with touch remoting transactions.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
...																protocolVersion															
...																															

header (6 bytes): An RDPINPUT_HEADER (section 2.2.2.6) structure. The **eventId** field MUST be set to EVENTID_SC_READY (0x0001).

protocolVersion (4 bytes): A 32-bit unsigned integer that specifies the input protocol version.

Value	Meaning
RDPINPUT_PROTOCOL_V100 0x00010000	Version 1.0.0 of the RDP input remoting protocol.
RDPINPUT_PROTOCOL_V101 0x00010001	Version 1.0.1 of the RDP input remoting protocol.

2.2.3.2 RDPINPUT_CS_READY_PDU

The **RDPINPUT_CS_READY_PDU** message is sent by the client endpoint and is used to indicate readiness to commence with touch remoting transactions.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
...																flags															
...																protocolVersion															
...																maxTouchContacts															

header (6 bytes): An RDPINPUT_HEADER (section 2.2.2.6) structure. The **eventId** field MUST be set to EVENTID_CS_READY (0x0002).

flags (4 bytes): A 32-bit unsigned integer that specifies touch initialization flags.

Flag	Meaning
READY_FLAGS_SHOW_TOUCH_VISUALS 0x00000001	Touch gesture and contact visuals SHOULD be rendered by the server in the remote session.
READY_FLAGS_DISABLE_TIMESTAMP_INJECTION 0x00000002	The client does not support touch frame timestamp remoting. The server MUST ignore any values specified in the **frameOffset** field of the RDPINPUT_TOUCH_FRAME (section 2.2.3.3.1) structure and the **encodeTime** field of the RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.3) message. This flag SHOULD NOT be sent to a server that only supports version 1.0.0 of the input remoting protocol. The server-supported version is specified in the **protocolVersion** field of the RDPINPUT_SC_READY_PDU (section 2.2.3.1) message.

protocolVersion (4 bytes): A 32-bit unsigned integer that specifies the input protocol version.

Value	Meaning
RDPINPUT_PROTOCOL_V100 0x00010000	Version 1.0.0 of the RDP input remoting protocol.
RDPINPUT_PROTOCOL_V101 0x00010001	Version 1.0.1 of the RDP input remoting protocol.

maxTouchContacts (2 bytes): A 16-bit unsigned integer that specifies the maximum number of simultaneous touch contacts supported by the client.

2.2.3.3 RDPINPUT_TOUCH_EVENT_PDU

The **RDPINPUT_TOUCH_EVENT_PDU** message is sent by the client endpoint and is used to remote a collection of touch frames.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
...																encodeTime (variable)															
...																															
frameCount (variable)																															
...																															
frames (variable)																															
...																															

header (6 bytes): An RDPINPUT_HEADER (section 2.2.2.6) structure. The **eventId** field MUST be set to EVENTID_TOUCH (0x0003).

encodeTime (variable): A FOUR_BYTE_UNSIGNED_INTEGER (section 2.2.2.3) structure that specifies the time that has elapsed (in milliseconds) from when the oldest touch frame was generated to when it was encoded for transmission by the client.

frameCount (variable): A TWO_BYTE_UNSIGNED_INTEGER (section 2.2.2.1) structure that specifies the number of RDPINPUT_TOUCH_FRAME (section 2.2.3.3.1) structures in the **frames** field.

frames (variable): An array of **RDPINPUT_TOUCH_FRAME** structures ordered from the oldest in time to the most recent in time. The number of structures in this array is specified by the **frameCount** field.

2.2.3.3.1 RDPINPUT_TOUCH_FRAME

The **RDPINPUT_TOUCH_FRAME** structure encapsulates a collection of RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structures that are part of the same logical touch frame.

Copyright © 2013 Microsoft Corporation.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
contactCount (variable)																															
...																															
frameOffset (variable)																															
...																															
contacts (variable)																															
...																															

contactCount (variable): A TWO_BYTE_UNSIGNED_INTEGER (section 2.2.2.1) structure that specifies the number of **RDPINPUT_CONTACT_DATA** structures in the **contacts** field.

frameOffset (variable): An EIGHT_BYTE_UNSIGNED_INTEGER (section 2.2.2.5) structure that specifies the time offset from the previous frame (in microseconds). If this is the first frame being transmitted then this field MUST be set to zero.

contacts (variable): An array of **RDPINPUT_CONTACT_DATA** structures. The number of structures in this array is specified by the **contactCount** field.

2.2.3.3.1.1 RDPINPUT_CONTACT_DATA

The **RDPINPUT_CONTACT_DATA** structure describes the characteristics of a contact that is encapsulated in an RDPINPUT_TOUCH_FRAME (section 2.2.3.3.1) structure.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
contactId								fieldsPresent (variable)																							
...																															
x (variable)																															
...																															
y (variable)																															
...																															
contactFlags (variable)																															
...																															

contactRectLeft (variable)
...
contactRectTop (variable)
...
contactRectRight (variable)
...
contactRectBottom (variable)
...
orientation (variable)
...
pressure (variable)
...

contactId (1 byte): An 8-bit unsigned integer that specifies the ID assigned to the contact. This value MUST be in the range 0x00 to 0xFF (inclusive).

fieldsPresent (variable): A TWO_BYTE_UNSIGNED_INTEGER (section 2.2.2.1) structure that specifies the presence of the optional **contactRectLeft**, **contactRectTop**, **contactRectRight**, **contactRectBottom**, **orientation**, and **pressure** fields.

Flag	Meaning
CONTACT_DATA_CONTACTRECT_PRESENT 0x0001	The optional **contactRectLeft**, **contactRectTop**, **contactRectRight**, and **contactRectBottom** fields are all present.
CONTACT_DATA_ORIENTATION_PRESENT 0x0002	The optional **orientation** field is present.
CONTACT_DATA_PRESSURE_PRESENT 0x0004	The optional **pressure** field is present.

x (variable): A FOUR_BYTE_SIGNED_INTEGER (section 2.2.2.4) structure that specifies the X-coordinate (relative to the virtual-desktop origin) of the contact.

y (variable): A **FOUR_BYTE_SIGNED_INTEGER** structure that specifies the Y-coordinate (relative to the virtual-desktop origin) of the contact.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

contactFlags (variable): A FOUR_BYTE_UNSIGNED_INTEGER (section 2.2.2.3) structure that specifies the current state of the contact.

Flag	Meaning
CONTACT_FLAG_DOWN 0x0001	The contact transitioned to the engaged state (made contact).
CONTACT_FLAG_UPDATE 0x0002	Contact update.
CONTACT_FLAG_UP 0x0004	The contact transitioned from the engaged state (broke contact).
CONTACT_FLAG_INRANGE 0x0008	The contact has not departed and is still in range.
CONTACT_FLAG_INCONTACT 0x0010	The contact is in the engaged state.
CONTACT_FLAG_CANCELED 0x0020	The contact has been canceled.

This field MUST contain one of the following combinations of the contact state flags and MUST NOT contain any other combination:

- UP
- UP | CANCELED
- UPDATE
- UPDATE | CANCELED
- DOWN | INRANGE | INCONTACT
- UPDATE | INRANGE | INCONTACT
- UP | INRANGE
- UPDATE | INRANGE

The figure "Lifetime of a touch contact" in section 3.1.1.1 describes the states through which a contact involved in a touch transaction can transition.

contactRectLeft (variable): An optional TWO_BYTE_SIGNED_INTEGER (section 2.2.2.2) structure that specifies the leftmost bound (relative to the contact point specified by the **x** and **y** fields) of the exclusive rectangle describing the geometry of the contact. This rectangle MUST be rotated counter-clockwise by the angle specified in the **orientation** field to yield the actual contact geometry. The presence of the **contactRectLeft** field is indicated by the CONTACT_DATA_CONTACTRECT_PRESENT (0x0001) flag in the **fieldsPresent** field.

contactRectTop (variable): An optional **TWO_BYTE_SIGNED_INTEGER** structure that specifies the upper bound (relative to the contact point specified by the **x** and **y** fields) of the exclusive rectangle describing the geometry of the contact. This rectangle MUST be rotated counter-clockwise by the angle specified in the **orientation** field to yield the actual contact

Copyright © 2013 Microsoft Corporation.

geometry. The presence of the **contactRectTop** field is indicated by the CONTACT_DATA_CONTACTRECT_PRESENT (0x0001) flag in the **fieldsPresent** field.

contactRectRight (variable): An optional **TWO_BYTE_SIGNED_INTEGER** structure that specifies the rightmost bound (relative to the contact point specified by the **x** and **y** fields) of the exclusive rectangle describing the geometry of the contact. This rectangle MUST be rotated counter-clockwise by the angle specified in the **orientation** field to yield the actual contact geometry. The presence of the **contactRectRight** field is indicated by the CONTACT_DATA_CONTACTRECT_PRESENT (0x0001) flag in the **fieldsPresent** field.

contactRectBottom (variable): An optional **TWO_BYTE_SIGNED_INTEGER** structure that specifies the lower bound (relative to the contact point specified by the **x** and **y** fields) of the exclusive rectangle describing the geometry of the contact. This rectangle MUST be rotated counter-clockwise by the angle specified in the **orientation** field to yield the actual contact geometry. The presence of the **contactRectBottom** field is indicated by the CONTACT_DATA_CONTACTRECT_PRESENT (0x0001) flag in the **fieldsPresent** field.

orientation (variable): An optional **FOUR_BYTE_UNSIGNED_INTEGER** structure that specifies the angle through which the contact rectangle (specified in the **contactRectLeft**, **contactRectTop**, **contactRectRight** and **contactRectBottom** fields) MUST be rotated to yield the actual contact geometry. This value MUST be in the range 0x00000000 to 0x00000167 (359), inclusive, where 0x00000000 indicates a touch contact aligned with the y-axis and pointing from bottom to top; increasing values indicate degrees of rotation in a counter-clockwise direction. The presence of the **orientation** field is indicated by the CONTACT_DATA_ORIENTATION_PRESENT (0x0002) flag in the **fieldsPresent** field. If the **orientation** field is not present the angle of rotation MUST be assumed to be zero degrees.

pressure (variable): An optional **FOUR_BYTE_UNSIGNED_INTEGER** structure that specifies the contact pressure. This value MUST be normalized in the range 0x00000000 to 0x0000FDE8 (65000), inclusive. The presence of this field is indicated by the CONTACT_DATA_PRESSURE_PRESENT (0x0004) flag in the **fieldsPresent** field.

2.2.3.4 RDPINPUT_SUSPEND_TOUCH_PDU

The **RDPINPUT_SUSPEND_TOUCH_PDU** message is sent by the server endpoint and is used to instruct the client to suspend the transmission of the RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.2) message.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
...																															

header (6 bytes): An RDPINPUT_HEADER (section 2.2.2.6) structure. The **eventId** field MUST be set to EVENTID_SUSPEND_TOUCH (0x0004).

2.2.3.5 RDPINPUT_RESUME_TOUCH_PDU

The **RDPINPUT_RESUME_TOUCH_PDU** message is sent by the server endpoint and is used to instruct the client to resume the transmission of the RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.2).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
...																															

header (6 bytes): An RDPINPUT_HEADER (section 2.2.2.6) structure. The **eventId** field MUST be set to EVENTID_RESUME_TOUCH (0x0005).

2.2.3.6 RDPINPUT_DISMISS_HOVERING_CONTACT_PDU

The **RDPINPUT_DISMISS_HOVERING_CONTACT_PDU** message is sent by the client endpoint to instruct the server to transition a contact in the "hovering" state to the "out of range" state (section 3.1.1.1).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
header																															
...																contactId															

header (6 bytes): An RDPINPUT_HEADER (section 2.2.2.6) structure. The **eventId** field MUST be set to EVENTID_DISMISS_HOVERING_CONTACT (0x0006).

contactId (1 byte): An 8-bit unsigned integer that specifies the ID assigned to the contact. This value MUST be in the range 0x00 to 0xFF (inclusive).

2.3 Directory Service Schema Elements

None.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

3.1.1.1 Touch Contact State Transitions

The following finite state machine diagram describes the states through which a contact involved in a touch transaction can transition during its lifetime.

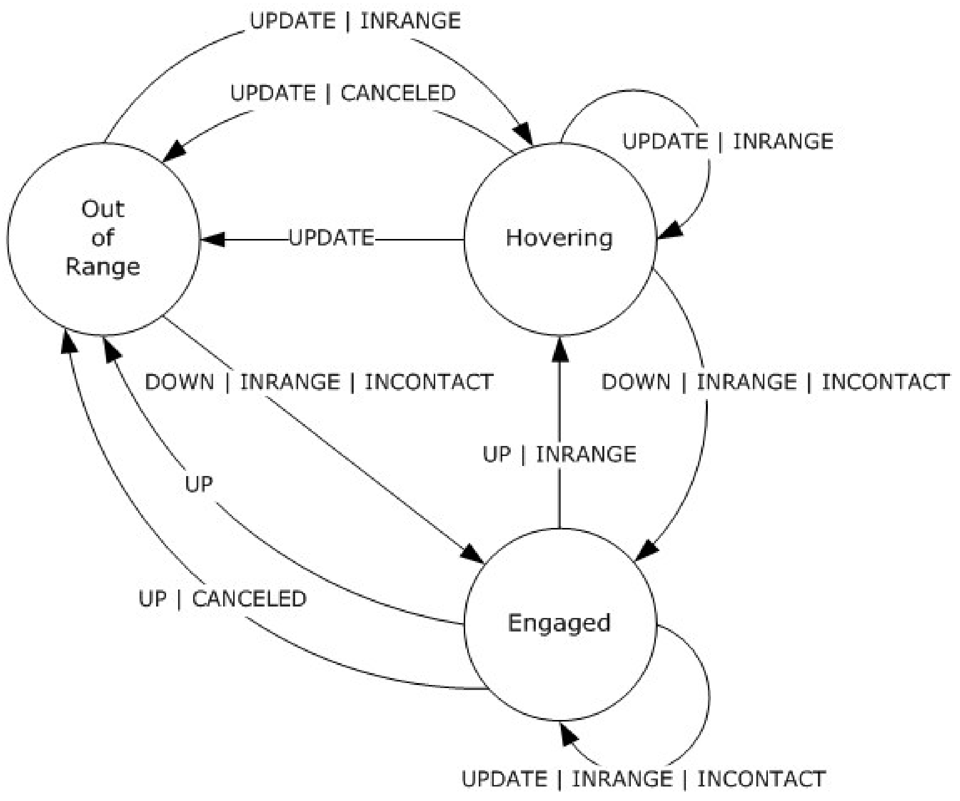


Figure 2: Lifetime of a touch contact

A touch contact transitions through three main states:

- Out of Range
- Hovering
- Engaged

When a contact is in the "hovering" or "engaged" state, it is referred to as being "active". "Hovering" contacts are in range of the digitizer, while "engaged" contacts are in range of the digitizer and in contact with the digitizer surface. The Remote Desktop Protocol: Input Virtual Channel Extension remotes only active contacts and contacts that are transitioning to the "out of range" state; see section 2.2.3.3.1.1 for an enumeration of valid state flags combinations.

When transitioning from the "engaged" state to the "hovering" state, or from the "engaged" state to the "out of range" state, the contact position cannot change; it is only allowed to change after the transition has taken place.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Processing an Input Message

All input messages are prefaced by the RDPINPUT_HEADER (section 2.2.2.6) structure.

When an input message is processed, the **eventId** field in the header MUST first be examined to determine if the message is within the subset of expected messages. If the message is not expected, it SHOULD be ignored.

If the message is in the correct sequence, the **pduLength** field MUST be examined to make sure that it is consistent with the amount of data read from the "Microsoft::Windows::RDS::Input" dynamic virtual channel (section 2.1). If this is not the case, the message SHOULD be ignored.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

The server MUST send the RDPINPUT_SC_READY_PDU (section 2.2.3.1) message to the client, as specified in section 3.2.5.1, to initiate the process of remoting touch input frames.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Sending an RDPINPUT_SC_READY_PDU Message

The structure and fields of the **RDPINPUT_SC_READY_PDU** message are specified in section 2.2.3.1.

If the server does not support touch injection, then it MUST NOT send this PDU to the client.

3.2.5.2 Processing an RDPINPUT_CS_READY_PDU Message

The structure and fields of the **RDPINPUT_CS_READY_PDU** message are specified in section 2.2.3.2.

The **header** field MUST be processed as specified in section 3.1.5.1. If the message is valid, the server SHOULD use the value specified by the client in the **maxTouchContacts** field to initialize the touch injection subsystem.

3.2.5.3 Processing an RDPINPUT_TOUCH_EVENT_PDU Message

The structure and fields of the **RDPINPUT_TOUCH_EVENT_PDU** message are specified in section 2.2.3.3.

The **header** field MUST be processed as specified in section 3.1.5.1. If the message is valid, the server MUST iterate over each RDPINPUT_TOUCH_FRAME (section 2.2.3.3.1) structure encapsulated in the **RDPINPUT_TOUCH_EVENT_PDU** message, decode each RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structure in the frame, and then inject the frame contacts into the user session.

If any of the contacts does not conform to the finite state machine described in section 3.1.1.1, the touch transaction SHOULD be canceled in the session, and all subsequent frames associated with the transaction SHOULD be ignored until a new touch transaction is initiated at the client.

3.2.5.4 Sending an RDPINPUT_SUSPEND_TOUCH_PDU message

The structure and fields of the **RDPINPUT_SUSPEND_TOUCH_PDU** message are specified in section 2.2.3.4.

To request that the client resume the transmission of touch frames, the server MUST send the RDPINPUT_RESUME_TOUCH_PDU (section 2.2.3.5) message to the client, as specified in section 3.2.5.5.

3.2.5.5 Sending an RDPINPUT_RESUME_TOUCH_PDU Message

The structure and fields of the **RDPINPUT_RESUME_TOUCH_PDU** message are specified in section 2.2.3.5.

Copyright © 2013 Microsoft Corporation.

The RDPINPUT_RESUME_TOUCH_PDU (section 2.2.3.5) message SHOULD be sent only if touch remoting has been suspended by using the RDPINPUT_SUSPEND_TOUCH_PDU (section 2.2.3.4) message, as specified in section 3.2.5.4.

3.2.5.6 Processing an RDPINPUT_DISMISS_HOVERING_CONTACT_PDU Message

The structure and fields of the **RDPINPUT_DISMISS_HOVERING_CONTACT_PDU** message are specified in section 2.2.3.6.

The **header** field MUST be processed as specified in section 3.1.5.1. If the message is valid, the server MUST transition the contact specified by the **contactId** field to the "out of range" state if it is in the hovering state. If no contact with the specified contact ID exists, or if the contact is in the engaged state, then no action MUST be taken.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Client Details

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of 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.

Note It is possible to implement the following conceptual data by using a variety of techniques as long as the implementation produces external behavior that is consistent with that described in this document.

3.3.1.1 Touch Remoting Suspended

The **Touch Remoting Suspended** abstract data model (ADM) element contains a Boolean value that indicates whether the capture, encoding, and transmission of touch frames on the client has been suspended. This value is toggled by the receipt of the RDPINPUT_SUSPEND_TOUCH_PDU (section 2.2.3.4) message, as specified in section 3.3.5.4, and the RDPINPUT_RESUME_TOUCH_PDU (section 2.2.3.5) message, as specified in section 3.3.5.5.

3.3.2 Timers

None.

3.3.3 Initialization

The client SHOULD NOT open the "Microsoft::Windows::RDS::Input" virtual channel transport (section 2.1) if a physical or virtual touch digitizer is not attached to the system.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

3.3.4 Higher-Layer Triggered Events

None.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Processing an RDPINPUT_SC_READY_PDU message

The structure and fields of the **RDPINPUT_SC_READY_PDU** message are specified in section 2.2.3.1.

The **header** field MUST be processed as specified in section 3.1.5.1. If the message is valid, the client SHOULD respond by sending the RDPINPUT_CS_READY_PDU (section 2.2.3.2) message to the server, as specified in section 3.3.5.2. After sending the **RDPINPUT_CS_READY_PDU** message to the server, the client SHOULD start remoting multitouch input frames by sending the RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.3) message to the server, as specified in section 3.3.5.3.

3.3.5.2 Sending an RDPINPUT_CS_READY_PDU message

The structure and fields of the **RDPINPUT_CS_READY_PDU** message are specified in section 2.2.3.2.

The client MUST populate the **maxTouchContacts** field to indicate the maximum number of touch contacts that can be active at any given point in time. This value is the sum of the maximum active contacts supported by each touch digitizer attached to the client.

3.3.5.3 Sending an RDPINPUT_TOUCH_EVENT_PDU message

The structure and fields of the **RDPINPUT_TOUCH_EVENT_PDU** message are specified in section 2.2.3.3.

Each RDPINPUT_TOUCH_EVENT_PDU (section 2.2.3.3) message contains an array of RDPINPUT_TOUCH_FRAME (section 2.2.3.3.1) structures, and each frame contains an array of RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structures. Every **RDPINPUT_CONTACT_DATA** structure represents the state and attributes of an active contact; see section 3.1.1.1 for a description of permissible touch contact state transitions.

Every touch frame received by the client from a touch digitizer MUST be encoded as an **RDPINPUT_TOUCH_FRAME** structure, the contacts being encoded as **RDPINPUT_CONTACT_DATA** structures. The number of encoded touch frames depends on the rate at which the digitizer generates touch frames. Once the touch frames have been encoded, they MUST be encapsulated in an **RDPINPUT_TOUCH_EVENT_PDU** message.

3.3.5.4 Processing an RDPINPUT_SUSPEND_TOUCH_PDU message

The structure and fields of the **RDPINPUT_SUSPEND_TOUCH_PDU** message are specified in section 2.2.3.4.

The **header** field MUST be processed as specified in section 3.1.5.1. If the message is valid, the client MUST set the Touch Remoting Suspended (section 3.3.1.1) ADM element to TRUE and MUST suspend the transmission of touch frames to the server. If the **Touch Remoting Suspended** ADM element is already set to TRUE, the client SHOULD ignore this message.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

3.3.5.5 Processing an RDPINPUT_RESUME_TOUCH_PDU message

The structure and fields of the **RDPINPUT_RESUME_TOUCH_PDU** message are specified in section 2.2.3.5.

The **header** field MUST be processed as specified in section 3.1.5.1. If the message is valid, the client SHOULD set the Touch Remoting Suspended (section 3.3.1.1) ADM element to FALSE and MUST resume the transmission of touch frames to the server. If the **Touch Remoting Suspended** ADM element is already set to FALSE, the client SHOULD ignore this message.

3.3.5.6 Sending an RDPINPUT_DISMISS_HOVERING_CONTACT_PDU message

The structure and fields of the **RDPINPUT_DISMISS_HOVERING_CONTACT_PDU** message are specified in section 2.2.3.6.

The **contactId** field MUST be initialized with the ID of a valid hovering contact that has to be transitioned to the "out of range" state.

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

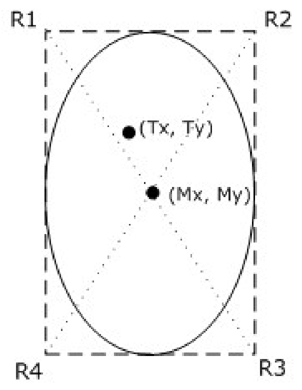
Copyright © 2013 Microsoft Corporation.

4 Protocol Examples

4.1 Contact Geometry Examples

The examples in sections 4.1.1 through to 4.1.4 present illustrations of touch contacts orientated at 0, 45, 90 and 315 degrees respectively. Based on the orientation of the contact, the contact geometry is rotated so that the height of the contact rectangle is parallel to the y-axis and the width parallel to the x-axis.

4.1.1 Touch Contact Oriented at 0 Degrees



In this case, the x, y, contact rectangle, and orientation of the RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structure are populated by using the following values:

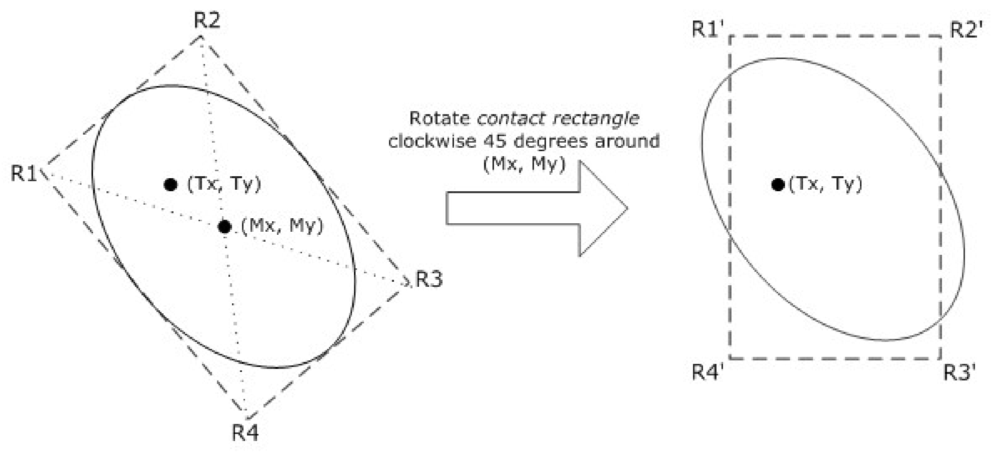
x = Tx

y = Ty

contact rectangle = (R1, R2, R3, R4)

orientation = 0 degrees

4.1.2 Touch Contact Oriented at 45 Degrees


R2
R1
(Tx, Ty)
(Mx, My)
R3
R4
Rotate *contact rectangle* clockwise 45 degrees around (Mx, My)
R1'
R2'
(Tx, Ty)
R4'
R3'

In this case, the x, y, contact rectangle, and orientation fields of the RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structure are populated by using the following values:

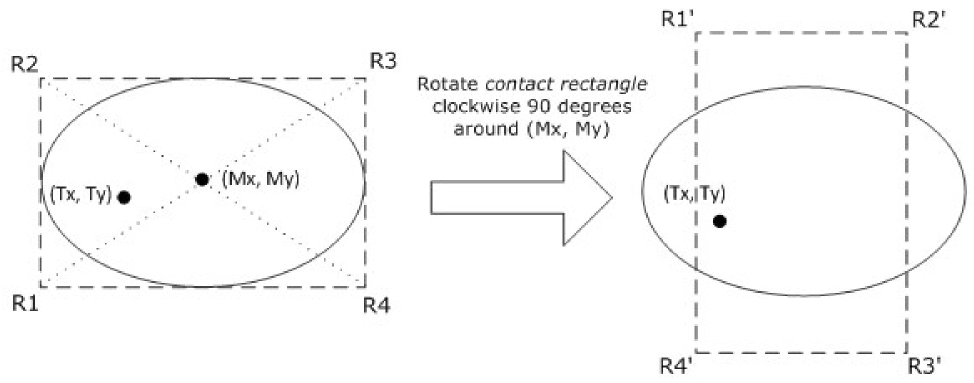
x = Tx

y = Ty

contact rectangle = (R1', R2', R3', R4')

orientation = 45 degrees

4.1.3 Touch Contact Oriented at 90 Degrees


R2
R3
R1
R4
(Tx, Ty)
(Mx, My)
Rotate *contact rectangle* clockwise 90 degrees around (Mx, My)
R1'
R2'
R4'
R3'
(Tx, Ty)

In this case, the x, y, contact rectangle, and orientation fields of the RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structure are populated by using the following values:

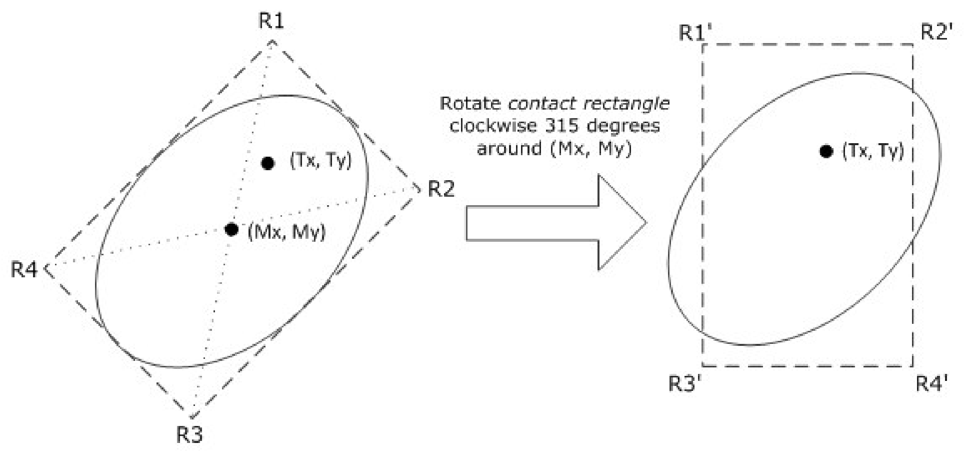
x = Tx

y = Ty

contact rectangle = (R1', R2', R3', R4')

orientation = 90 degrees

4.1.4 Touch Contact Oriented at 315 Degrees


R1
R2
R3
R4
(Tx, Ty)
(Mx, My)
Rotate *contact rectangle* clockwise 315 degrees around (Mx, My)
R1'
R2'
R3'
R4'
(Tx, Ty)

In this case, the x, y, contact rectangle, and orientation fields of the RDPINPUT_CONTACT_DATA (section 2.2.3.3.1.1) structure are populated by using the following values:

x = Tx

y = Ty

contactRect = (R1', R2', R3', R4')

orientation = 315 degrees

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

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 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.

Copyright © 2013 Microsoft Corporation.

7 Change Tracking

This section identifies changes that were made to the [MS-RDPEI] protocol document between the January 2013 and August 2013 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.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

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 language and 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 or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

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.

Copyright © 2013 Microsoft Corporation.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- 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 protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.3.1 RDPINPUT_SC_READY_PDU	Added version details.	N	Content updated.
2.2.3.2 RDPINPUT_CS_READY_PDU	Added version information for the READY_FLAGS_DISABLE_TIMESTAMP_INJECTION flag.	N	Content updated.
3.1.5.1 Processing an Input Message	67785 Changed "connection SHOULD be dropped" to "message SHOULD be ignored".	N	Content updated.

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

8 Index

Copyright © 2013 Microsoft Corporation.

Release: Monday, July 22, 2013

Copyright © 2013 Microsoft Corporation.