

## [MS-RDPESC-Diff]:

# Remote Desktop Protocol: Smart Card Virtual Channel Extension

---

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation (“this documentation”) for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **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 can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft [Open Specifications Promise](#) or the [Microsoft Community Promise](#). If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting [iplg@microsoft.com](mailto:iplg@microsoft.com).
- **License Programs.** To see all of the protocols in scope under a specific license program and the associated patents, visit the [Patent Map](#).
- **Trademarks.** The names of companies and products contained in this documentation might 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](http://www.microsoft.com/trademarks).
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are 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 as specifically described above, whether by implication, estoppel, or otherwise.

**Tools.** The Open Specifications documentation does 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 documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

**Support.** For questions and support, please contact [dochelp@microsoft.com](mailto:dochelp@microsoft.com).

## Revision Summary

Date	Revision History	Revision Class	Comments
6/1/2007	1.0	Major	Updated and revised the technical content.
7/3/2007	1.0.1	Editorial	Changed language and formatting in the technical content.
7/20/2007	1.0.2	Editorial	Changed language and formatting in the technical content.
8/10/2007	1.0.3	Editorial	Changed language and formatting in the technical content.
9/28/2007	1.0.4	Editorial	Changed language and formatting in the technical content.
10/23/2007	1.0.5	Editorial	Changed language and formatting in the technical content.
11/30/2007	2.0	Major	Normative reference.
1/25/2008	2.0.1	Editorial	Changed language and formatting in the technical content.
3/14/2008	2.0.2	Editorial	Changed language and formatting in the technical content.
5/16/2008	2.0.3	Editorial	Changed language and formatting in the technical content.
6/20/2008	2.0.4	Editorial	Changed language and formatting in the technical content.
7/25/2008	2.0.5	Editorial	Changed language and formatting in the technical content.
8/29/2008	2.0.6	Editorial	Changed language and formatting in the technical content.
10/24/2008	2.0.7	Editorial	Changed language and formatting in the technical content.
12/5/2008	3.0	Major	Updated and revised the technical content.
1/16/2009	3.0.1	Editorial	Changed language and formatting in the technical content.
2/27/2009	3.0.2	Editorial	Changed language and formatting in the technical content.
4/10/2009	3.0.3	Editorial	Changed language and formatting in the technical content.
5/22/2009	4.0	Major	Updated and revised the technical content.
7/2/2009	4.0.1	Editorial	Changed language and formatting in the technical content.
8/14/2009	4.0.2	Editorial	Changed language and formatting in the technical content.
9/25/2009	4.1	Minor	Clarified the meaning of the technical content.
11/6/2009	4.1.1	Editorial	Changed language and formatting in the technical content.
12/18/2009	5.0	Major	Updated and revised the technical content.
1/29/2010	5.1	Minor	Clarified the meaning of the technical content.
3/12/2010	6.0	Major	Updated and revised the technical content.
4/23/2010	7.0	Major	Updated and revised the technical content.
6/4/2010	8.0	Major	Updated and revised the technical content.
7/16/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.

<b>Date</b>	<b>Revision History</b>	<b>Revision Class</b>	<b>Comments</b>
8/27/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	8.0	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	8.0	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	8.1	Minor	Clarified the meaning of the technical content.
9/23/2011	8.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	9.0	Major	Updated and revised the technical content.
3/30/2012	9.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	9.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	9.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	9.0	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	10.0	Major	Updated and revised the technical content.
11/14/2013	10.0	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	10.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	10.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	11.0	Major	Significantly changed the technical content.
10/16/2015	11.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	11.0.1	Editorial	Changed language and formatting in the technical content.
6/1/2017	12.0	Major	Significantly changed the technical content.
9/15/2017	13.0	Major	Significantly changed the technical content.

Date	Revision History	Revision Class	Comments
<u>12/1/2017</u>	<u>13.0</u>	<u>None</u>	<u>No changes to the meaning, language, or formatting of the technical content.</u>

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>10</b>
1.1	Glossary .....	10
1.2	References .....	12
1.2.1	Normative References .....	12
1.2.2	Informative References .....	13
1.3	Overview .....	13
1.4	Relationship to Other Protocols .....	15
1.5	Prerequisites/Preconditions .....	15
1.6	Applicability Statement .....	16
1.7	Versioning and Capability Negotiation .....	16
1.8	Vendor-Extensible Fields .....	16
1.9	Standards Assignments.....	16
<b>2</b>	<b>Messages.....</b>	<b>18</b>
2.1	Transport .....	18
2.2	Common Data Types .....	18
2.2.1	Common Structures .....	18
2.2.1.1	REDIR_SCARDCONTEXT.....	18
2.2.1.2	REDIR_SCARDHANDLE .....	18
2.2.1.3	Connect_Common.....	19
2.2.1.4	LocateCards_ATRMask.....	19
2.2.1.5	ReaderState_Common_Call .....	19
2.2.1.6	ReaderStateA .....	20
2.2.1.7	ReaderStateW .....	20
2.2.1.8	SCardIO_Request .....	20
2.2.1.9	ReadCache_Common.....	21
2.2.1.10	WriteCache_Common .....	21
2.2.1.11	ReaderState_Return .....	21
2.2.2	TS Server-Generated Structures.....	22
2.2.2.1	EstablishContext_Call .....	22
2.2.2.2	Context_Call.....	22
2.2.2.3	ListReaderGroups_Call.....	23
2.2.2.4	ListReaders_Call .....	23
2.2.2.5	ContextAndStringA_Call.....	24
2.2.2.6	ContextAndStringW_Call .....	24
2.2.2.7	ContextAndTwoStringA_Call .....	25
2.2.2.8	ContextAndTwoStringW_Call .....	26
2.2.2.9	LocateCardsA_Call .....	26
2.2.2.10	LocateCardsW_Call.....	27
2.2.2.11	GetStatusChangeA_Call .....	27
2.2.2.12	GetStatusChangeW_Call .....	27
2.2.2.13	ConnectA_Call .....	28
2.2.2.14	ConnectW_Call .....	28
2.2.2.15	Reconnect_Call .....	28
2.2.2.16	HCardAndDisposition_Call .....	29
2.2.2.17	State_Call .....	30
2.2.2.18	Status_Call .....	30
2.2.2.19	Transmit_Call .....	31
2.2.2.20	Control_Call .....	31
2.2.2.21	GetAttrib_Call.....	32
2.2.2.22	SetAttrib_Call .....	33
2.2.2.23	LocateCardsByATRA_Call .....	33
2.2.2.24	LocateCardsByATRW_Call .....	34
2.2.2.25	ReadCacheA_Call .....	34
2.2.2.26	ReadCacheW_Call .....	34

2.2.2.27	WriteCacheA_Call.....	34
2.2.2.28	WriteCacheW_Call.....	35
2.2.2.29	GetTransmitCount_Call.....	35
2.2.2.30	ScardAccessStartedEvent_Call.....	35
2.2.2.31	GetReaderIcon_Call.....	35
2.2.2.32	GetDeviceTypeId_Call.....	36
2.2.3	TS Client-Generated Structures.....	36
2.2.3.1	ReadCache_Return.....	36
2.2.3.2	EstablishContext_Return.....	36
2.2.3.3	Long_Return.....	37
2.2.3.4	ListReaderGroups_Return and ListReaders_Return.....	37
2.2.3.5	LocateCards_Return and GetStatusChange_Return.....	37
2.2.3.6	Control_Return.....	38
2.2.3.7	Reconnect_Return.....	38
2.2.3.8	Connect_Return.....	38
2.2.3.9	State_Return.....	39
2.2.3.10	Status_Return.....	39
2.2.3.11	Transmit_Return.....	40
2.2.3.12	GetAttrib_Return.....	40
2.2.3.13	GetTransmitCount_Return.....	41
2.2.3.14	GetReaderIcon_Return.....	41
2.2.3.15	GetDeviceTypeId_Return.....	41
2.2.4	Card/Reader State.....	41
2.2.5	Protocol Identifier.....	42
2.2.6	Access Mode Flags.....	43
2.2.7	Reader State.....	43
2.2.8	Return Code.....	45
<b>3</b>	<b>Protocol Details.....</b>	<b>49</b>
3.1	Protocol Server Details.....	49
3.1.1	Abstract Data Model.....	49
3.1.2	Timers.....	49
3.1.3	Initialization.....	49
3.1.4	Message Processing Events and Sequencing Rules.....	49
3.1.4.1	SCARD_IOCTL_ESTABLISHCONTEXT (IOCTL 0x00090014).....	53
3.1.4.2	SCARD_IOCTL_RELEASECONTEXT (IOCTL 0x00090018).....	53
3.1.4.3	SCARD_IOCTL_ISVALIDCONTEXT (IOCTL 0x0009001C).....	53
3.1.4.4	SCARD_IOCTL_ACCESSSTARTEDEVENT (IOCTL 0x000900E0).....	53
3.1.4.5	SCARD_IOCTL_LISTREADERGROUPSA (IOCTL 0x00090020).....	54
3.1.4.6	SCARD_IOCTL_LISTREADERGROUPSW (IOCTL 0x00090024).....	54
3.1.4.7	SCARD_IOCTL_LISTREADERSA (IOCTL 0x00090028).....	54
3.1.4.8	SCARD_IOCTL_LISTREADERSW (IOCTL 0x0009002C).....	54
3.1.4.9	SCARD_IOCTL_INTRODUCEREADERGROUPA (IOCTL 0x00090050).....	54
3.1.4.10	SCARD_IOCTL_INTRODUCEREADERGROUPW (IOCTL 0x00090054).....	55
3.1.4.11	SCARD_IOCTL_FORGETREADERGROUPA (IOCTL 0x00090058).....	55
3.1.4.12	SCARD_IOCTL_FORGETREADERGROUPW (IOCTL 0x0009005C).....	55
3.1.4.13	SCARD_IOCTL_INTRODUCEREADERA (IOCTL 0x00090060).....	55
3.1.4.14	SCARD_IOCTL_INTRODUCEREADERW (IOCTL 0x00090064).....	55
3.1.4.15	SCARD_IOCTL_FORGETREADERA (IOCTL 0x00090068).....	55
3.1.4.16	SCARD_IOCTL_FORGETREADERW (IOCTL 0x0009006C).....	56
3.1.4.17	SCARD_IOCTL_ADDREADERTOGROUPA (IOCTL 0x00090070).....	56
3.1.4.18	SCARD_IOCTL_ADDREADERTOGROUPW (IOCTL 0x00090074).....	56
3.1.4.19	SCARD_IOCTL_REMOVEREADERFROMGROUPA (IOCTL 0x00090078).....	56
3.1.4.20	SCARD_IOCTL_REMOVEREADERFROMGROUPW (IOCTL 0x0009007C).....	56
3.1.4.21	SCARD_IOCTL_LOCATECARDSA (IOCTL 0x00090098).....	56
3.1.4.22	SCARD_IOCTL_LOCATECARDSW (IOCTL 0x0009009C).....	57
3.1.4.23	SCARD_IOCTL_GETSTATUSCHANGEA (IOCTL 0x000900A0).....	57
3.1.4.24	SCARD_IOCTL_GETSTATUSCHANGEW (IOCTL 0x000900A4).....	57

3.1.4.25	SCARD_IOCTL_LOCATECARDSBYATRA (IOCTL 0x000900E8)	57
3.1.4.26	SCARD_IOCTL_LOCATECARDSBYATRW (IOCTL 0x000900EC)	57
3.1.4.27	SCARD_IOCTL_CANCEL (IOCTL 0x000900A8)	58
3.1.4.28	SCARD_IOCTL_CONNECTA (IOCTL 0x000900AC)	58
3.1.4.29	SCARD_IOCTL_CONNECTW (IOCTL 0x000900B0)	58
3.1.4.30	SCARD_IOCTL_DISCONNECT (IOCTL 0x000900B8)	58
3.1.4.31	SCARD_IOCTL_BEGINTRANSACTION (IOCTL 0x000900BC)	58
3.1.4.32	SCARD_IOCTL_ENDTRANSACTION (IOCTL 0x000900C0)	58
3.1.4.33	SCARD_IOCTL_STATUSA (IOCTL 0x000900C8)	59
3.1.4.34	SCARD_IOCTL_STATUSW (IOCTL 0x000900CC)	59
3.1.4.35	SCARD_IOCTL_TRANSMIT (IOCTL 0x000900D0)	59
3.1.4.36	SCARD_IOCTL_RECONNECT (IOCTL 0x000900B4)	59
3.1.4.37	SCARD_IOCTL_CONTROL (IOCTL 0x000900D4)	59
3.1.4.38	SCARD_IOCTL_GETATTRIB (IOCTL 0x000900D8)	59
3.1.4.39	SCARD_IOCTL_SETATTRIB (IOCTL 0x000900DC)	60
3.1.4.40	SCARD_IOCTL_STATE (IOCTL 0x000900C4)	60
3.1.4.41	SCARD_IOCTL_GETTRANSMITCOUNT (IOCTL 0x00090100)	60
3.1.4.42	SCARD_IOCTL_READCACHEA (IOCTL 0x000900F0)	60
3.1.4.43	SCARD_IOCTL_READCACHEW (IOCTL 0x000900F4)	60
3.1.4.44	SCARD_IOCTL_WRITECACHEA (IOCTL 0x000900F8)	61
3.1.4.45	SCARD_IOCTL_WRITECACHEW (IOCTL 0x000900FC)	61
3.1.4.46	SCARD_IOCTL_RELEASETARTEDEVENT	61
3.1.4.47	SCARD_IOCTL_GETREADERICON (IOCTL 0x00090104)	61
3.1.4.48	SCARD_IOCTL_GETDEVICETYPEID (IOCTL 0x00090108)	61
3.1.5	Timer Events	61
3.1.6	Other Local Events	61
3.2	Protocol Client Details	62
3.2.1	Abstract Data Model	62
3.2.2	Timers	62
3.2.3	Initialization	62
3.2.4	Higher-Layer Triggered Events	62
3.2.5	Message Processing Events and Sequencing Rules	62
3.2.5.1	Sending Outgoing Messages	62
3.2.5.2	Processing Incoming Replies	62
3.2.5.3	Messages	63
3.2.5.3.1	Sending EstablishContext Message	63
3.2.5.3.2	Processing EstablishContext Reply	63
3.2.5.3.3	Sending ReleaseContext Message	63
3.2.5.3.4	Processing ReleaseContext Reply	63
3.2.5.3.5	Sending IntroduceReader (ASCII) Message	63
3.2.5.3.6	Processing IntroduceReader (ASCII) Reply	63
3.2.5.3.7	Sending IntroduceReader (Unicode) Message	63
3.2.5.3.8	Processing IntroduceReader (Unicode) Reply	63
3.2.5.3.9	Sending ForgetReader (ASCII) Message	63
3.2.5.3.10	Processing ForgetReader (ASCII) Reply	63
3.2.5.3.11	Sending ForgetReader (Unicode) Message	64
3.2.5.3.12	Processing ForgetReader (Unicode) Reply	64
3.2.5.3.13	Sending IntroduceReaderGroup (ASCII) Message	64
3.2.5.3.14	Processing IntroduceReaderGroup (ASCII) Reply	64
3.2.5.3.15	Sending IntroduceReaderGroup (Unicode) Message	64
3.2.5.3.16	Processing IntroduceReaderGroup (Unicode) Reply	64
3.2.5.3.17	Sending ForgetReaderGroup (ASCII) Message 1	64
3.2.5.3.18	Processing ForgetReaderGroup (ASCII) Reply	64
3.2.5.3.19	Sending ForgetReaderGroup (ASCII) Message 2	64
3.2.5.3.20	Processing ForgetReaderGroup (Unicode) Reply	64
3.2.5.3.21	Sending AddReaderToGroup (ASCII) Message	64
3.2.5.3.22	Processing AddReaderToGroup (ASCII) Reply	65
3.2.5.3.23	Sending AddReaderToGroup (Unicode) Message	65

3.2.5.3.24	Processing AddReaderToGroup (Unicode) Reply.....	65
3.2.5.3.25	Sending RemoveReaderFromGroup (ASCII) Message .....	65
3.2.5.3.26	Processing RemoveReaderFromGroup (ASCII) Reply .....	65
3.2.5.3.27	Sending RemoveReaderFromGroup (Unicode) Message .....	65
3.2.5.3.28	Processing RemoveReaderFromGroup (Unicode) Reply .....	65
3.2.5.3.29	Sending ListReaderGroups (ASCII) Message .....	65
3.2.5.3.30	Processing ListReaderGroups (ASCII) Reply .....	65
3.2.5.3.31	Sending ListReaderGroups (Unicode) Message .....	65
3.2.5.3.32	Processing ListReaderGroups (Unicode) Reply .....	65
3.2.5.3.33	Sending ListReaders (ASCII) Message .....	66
3.2.5.3.34	Processing ListReadersReply (ASCII) Reply .....	66
3.2.5.3.35	Sending ListReaders (Unicode) Message .....	66
3.2.5.3.36	Processing ListReadersReply (Unicode) Reply .....	66
3.2.5.3.37	Sending LocateCards (ASCII) Message .....	66
3.2.5.3.38	Processing LocateCards (ASCII) Reply .....	66
3.2.5.3.39	Sending LocateCards (Unicode) Message .....	66
3.2.5.3.40	Processing LocateCards (Unicode) Reply .....	66
3.2.5.3.41	Sending GetStatusChange (ASCII) Message.....	66
3.2.5.3.42	Processing GetStatusChange (ASCII) Reply .....	66
3.2.5.3.43	Sending GetStatusChange (Unicode) Message.....	66
3.2.5.3.44	Processing GetStatusChange (Unicode) Reply .....	67
3.2.5.3.45	Sending Cancel Message .....	67
3.2.5.3.46	Processing Cancel Reply .....	67
3.2.5.3.47	Sending Connect (ASCII) Message.....	67
3.2.5.3.48	Processing Connect (ASCII) Reply .....	67
3.2.5.3.49	Sending Connect (Unicode) Message.....	67
3.2.5.3.50	Processing Connect (Unicode) Reply .....	67
3.2.5.3.51	Sending Reconnect Message .....	67
3.2.5.3.52	Processing Reconnect Reply .....	67
3.2.5.3.53	Sending Disconnect Message .....	67
3.2.5.3.54	Processing Disconnect Reply .....	67
3.2.5.3.55	Sending Status (ASCII) Message .....	68
3.2.5.3.56	Processing Status (ASCII) Reply .....	68
3.2.5.3.57	Sending Status (Unicode) Message .....	68
3.2.5.3.58	Processing Status (Unicode) Reply .....	68
3.2.5.3.59	Sending State Message .....	68
3.2.5.3.60	Processing State Message Reply .....	68
3.2.5.3.61	Sending BeginTransaction Message.....	68
3.2.5.3.62	Processing BeginTransaction Reply .....	68
3.2.5.3.63	Sending EndTransaction Message .....	68
3.2.5.3.64	Processing EndTransaction Reply .....	68
3.2.5.3.65	Sending Transmit Message .....	68
3.2.5.3.66	Processing Transmit Reply .....	69
3.2.5.3.67	Sending Control Message .....	69
3.2.5.3.68	Processing Control Reply .....	69
3.2.5.3.69	Sending GetReaderCapabilities Message .....	69
3.2.5.3.70	Processing GetReaderCapabilities Reply.....	69
3.2.5.3.71	Sending SetReaderCapabilities Message .....	69
3.2.5.3.72	Processing SetReaderCapabilities Reply .....	69
3.2.5.3.73	Sending WaitForResourceManager Message .....	69
3.2.5.3.74	Processing WaitForResourceManager Reply.....	69
3.2.5.3.75	Sending LocateCardsByATR (ASCII) Message .....	69
3.2.5.3.76	Processing LocateCardsByATR (Unicode) Reply.....	69
3.2.5.3.77	Processing LocateCardsByATR (ASCII) Reply.....	69
3.2.5.3.78	Sending LocateCardsByATR (Unicode) Message .....	70
3.2.5.3.79	Sending ReadCache (ASCII) Message.....	70
3.2.5.3.80	Processing ReadCache (ASCII) Reply .....	70
3.2.5.3.81	Sending ReadCache (Unicode) Message.....	70



3.2.5.3.82	Processing ReadCache (Unicode) Reply .....	70
3.2.5.3.83	Sending WriteCache (ASCII) Message .....	70
3.2.5.3.84	Processing WriteCache (ASCII) Reply .....	70
3.2.5.3.85	Sending WriteCache (Unicode) Message .....	70
3.2.5.3.86	Processing WriteCache (Unicode) Reply .....	70
3.2.5.3.87	Sending GetTransmitCount Message .....	70
3.2.5.3.88	Processing GetTransmitCount Reply .....	70
3.2.5.3.89	Sending GetReaderIcon Message .....	71
3.2.5.3.90	Processing GetReaderIcon Reply .....	71
3.2.5.3.91	Sending GetDeviceTypeId Message .....	71
3.2.5.3.92	Processing GetDeviceTypeId Reply .....	71
3.2.6	Timer Events .....	71
3.2.7	Other Local Events .....	71
<b>4</b>	<b>Protocol Examples .....</b>	<b>72</b>
4.1	Establish Context Call .....	73
4.2	Establish Context Return .....	73
4.3	List Readers Call .....	73
4.4	List Readers Return .....	73
4.5	Get Status Change Call .....	73
4.6	Get Status Change Return .....	74
4.7	Connect Call .....	74
4.8	Connect Return .....	74
4.9	Begin Transaction Call .....	74
4.10	Begin Transaction Return .....	75
4.11	Status Call .....	75
4.12	Status Return .....	75
4.13	End Transaction Call .....	75
4.14	End Transaction Return .....	75
4.15	Disconnect Call .....	75
4.16	Disconnect Return .....	76
4.17	Release Context Call .....	76
4.18	Release Context Return .....	76
<b>5</b>	<b>Security .....</b>	<b>77</b>
5.1	Security Considerations for Implementers .....	77
5.2	Index of Security Parameters .....	77
<b>6</b>	<b>Appendix A: Full IDL .....</b>	<b>78</b>
<b>7</b>	<b>Appendix B: Product Behavior .....</b>	<b>84</b>
<b>8</b>	<b>Change Tracking .....</b>	<b>85</b>
<b>9</b>	<b>Index .....</b>	<b>86</b>

# 1 Introduction

This document specifies an extension (including virtual channels) to the Remote Desktop Protocol: File System Virtual Channel Extension for supporting smart card reader-like devices.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

## 1.1 Glossary

This document uses the following terms:

**Answer To Reset (ATR):** The transmission sent by an ISO-7816-compliant Integrated Circuit Card (as specified in [ISO/IEC-7816-3] section 8) to a smart card reader in response to an ISO-7816-3-based RESET condition.

**ASCII:** The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

**build number:** A part of a sequential numbering system that is used to differentiate one version of a software product from another.

**call packet:** A combination of I/O control (IOCTL) and a data structure request from a protocol client that corresponds to that IOCTL.

**card type:** A string that specifies a specific type of smart card that is recognized by Smart Cards for Windows.

**device:** Any peripheral or part of a computer system that can send or receive data.

**device I/O:** Device input/output.

**device name:** The friendly, human-readable name of a device.

**HRESULT:** An integer value that indicates the result or status of an operation. A particular HRESULT can have different meanings depending on the protocol using it. See [MS-ERREF] section 2.1 and specific protocol documents for further details.

**I/O control (IOCTL):** A command that is issued to a target file system or target device in order to query or alter the behavior of the target; or to query or alter the data and attributes that are associated with the target or the objects that are exposed by the target.

**Interface Definition Language (IDL):** The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

**Microsoft Terminal Services (TS):** A component that allows a user to access applications or data stored on a remote computer over a network connection.

**Multistring:** A series of null-terminated character strings terminated by a final null character stored in a contiguous block of memory.

**operating system version:** A uniquely identifiable numbered string that is used to identify a particular operating system.

**protocol client:** An endpoint that initiates a protocol.

**protocol server:** An endpoint that processes the call packet from a protocol client.

**reader group name:** The friendly, human-readable name for a reader group.

**Remote Desktop Protocol (RDP):** A multi-channel protocol that allows a user to connect to a computer running Microsoft Terminal Services (TS). RDP enables the exchange of client and server settings and also enables negotiation of common settings to use for the duration of the connection, so that input, graphics, and other data can be exchanged and processed between client and server.

**remote procedure call (RPC):** ~~A context-dependent communication protocol used primarily between client and server. The term commonly overloaded with has three meanings. Noted definitions that much of the industry literature concerning RPC technologies uses this term are often used interchangeably for any of the three meanings. Following are the three definitions: (\*) The: a runtime environment providing remote procedure call for communication facilities. The preferred usage for this meaning is "between computers (the RPC runtime". (\*) The pattern); a set of request-and-response message exchanges between computers (the RPC exchange-between-two-parties (typically, a client and a server). The preferred usage for this meaning is "RPC exchange". (\*) A-); and the single message from an RPC exchange as defined in (the previous definition. The preferred usage for this term is "RPC message". For more information about). The RPC, see specification is [C706].~~

**return packet:** An encoded structure containing the result of a call packet operation executed on the protocol client.

**smart card:** A portable device that is shaped like a business card and is embedded with a memory chip and either a microprocessor or some non-programmable logic. Smart cards are often used as authentication tokens and for secure key storage. Smart cards used for secure key storage have the ability to perform cryptographic operations with the stored key without allowing the key itself to be read or otherwise extracted from the card.

**smart card reader:** A device used as a communication medium between the smart card and a Host; for example, a computer. Also referred to as a Reader.

**smart card reader name:** The friendly, human-readable name of the smart card reader. Also referred to as a Reader Name.

**Smart Cards for Windows:** An implementation of the ICC Resource Manager according to [PCSC5].

**static virtual channel:** A static transport used for lossless communication between a client component and a server component over a main data connection, as specified in [MS-RDPBCGR].

**TS client:** A Microsoft Terminal Services program that initiates a connection.

**TS server:** A Microsoft Terminal Services program that responds to a request from a TS client.

**Unicode:** A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The Unicode standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

**Unicode string:** A Unicode 8-bit string is an ordered sequence of 8-bit units, a Unicode 16-bit string is an ordered sequence of 16-bit code units, and a Unicode 32-bit string is an ordered sequence of 32-bit code units. In some cases, it could be acceptable not to terminate with a terminating null character. Unless otherwise specified, all Unicode strings follow the UTF-16LE encoding scheme with no Byte Order Mark (BOM).

**universally unique identifier (UUID):** A 128-bit value. UUIDs can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects in cross-process communication such as client and server interfaces, manager entry-point vectors, and RPC objects. UUIDs are highly likely to be unique. UUIDs are also known as globally unique identifiers (GUIDs) and these terms are used interchangeably in the Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the UUID. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the UUID.

**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 defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

## 1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

### 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.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, <https://www2.opengroup.org/ogsys/catalog/c706>

[ISO/IEC-7816-3] International Organization for Standardization, "Identification Cards -- Integrated Circuit Cards -- Part 3: Cards with Contacts -- Electrical Interface and Transmission Protocols", ISO/IEC 7816-3, October 2006, [http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=38770](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=38770)

**Note** There is a charge to download the specification.

[ISO/IEC-7816-4] International Organization for Standardization, "Identification Cards -- Integrated Circuit Cards -- Part 4: Organization, Security, and Commands for Interchange", ISO/IEC 7816-4, January 2005, [http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=36134](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=36134)

**Note** There is a charge to download the specification.

[MS-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol".

[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".

[MS-RDPEFS] Microsoft Corporation, "Remote Desktop Protocol: File System Virtual Channel Extension".

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions".

[PCSC3] PC/SC Workgroup, "Interoperability Specification for ICCs and Personal Computer Systems - Part 3: Requirements for PC-Connected Interface Devices", December 1997, <http://www.pcscworkgroup.com/specifications/V1/p3v10doc>

[PCSC5] PC/SC Workgroup, "Interoperability Specification for ICCs and Personal Computer Systems - Part 5: ICC Resource Manager Definition", December 1997, <http://www.pcscworkgroup.com/specifications/V1/p5v10doc>

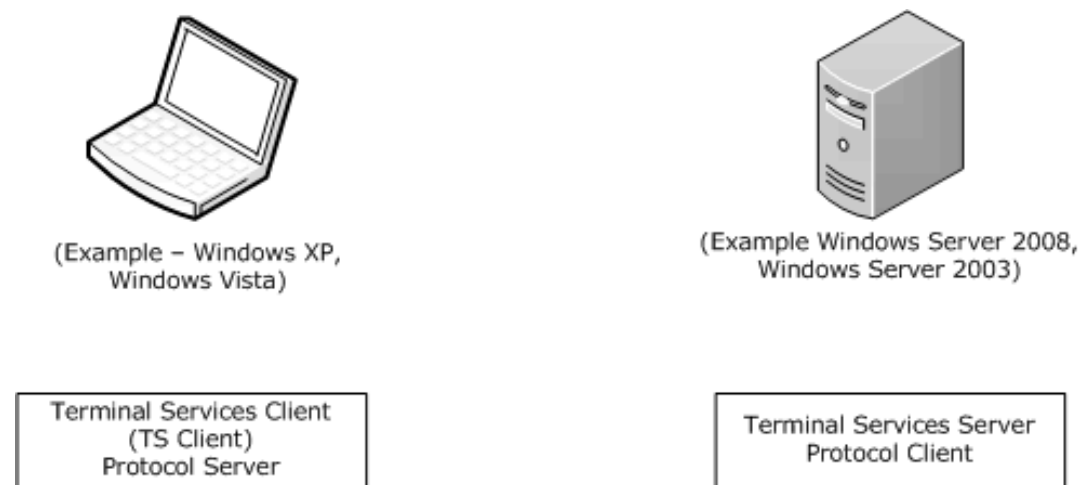
[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.rfc-editor.org/rfc/rfc2119.txt>

## 1.2.2 Informative References

None.

## 1.3 Overview

The following figure illustrates a baseline for terminology related to clients and servers.



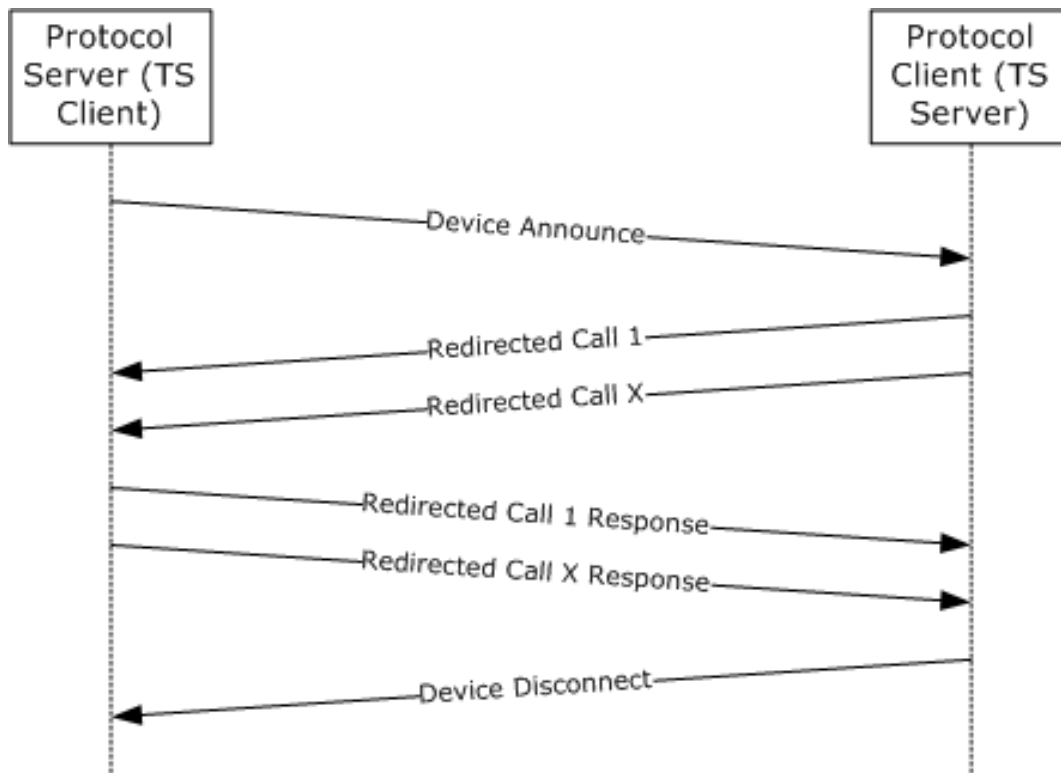
**Figure 1: TS and protocol client-server definition**

Remote Desktop Protocol (RDP) Device Redirection enables client devices (for example, printers, smart card readers, drives, audio, serial ports, and parallel ports) to be available to server-side applications, within the context of a single RDP session. This protocol is specified in [MS-RDPEFS].

Smart Card Redirection is an asynchronous client/server protocol, an extension (specified in [MS-RDPEFS]) that is designed to remotely execute requests on a client's Smart Cards for Windows. These requests would have otherwise been executed on the server. Each request is composed of two packets: a call packet and return packet. The protocol client (Microsoft Terminal Services (TS) server) sends a call packet after an initial announcement by the protocol server (TS client), and will receive a return packet after the request has been completed or an error has occurred. Remote Desktop Protocol (RDP) Device Redirection uses a static virtual channel as its transport.

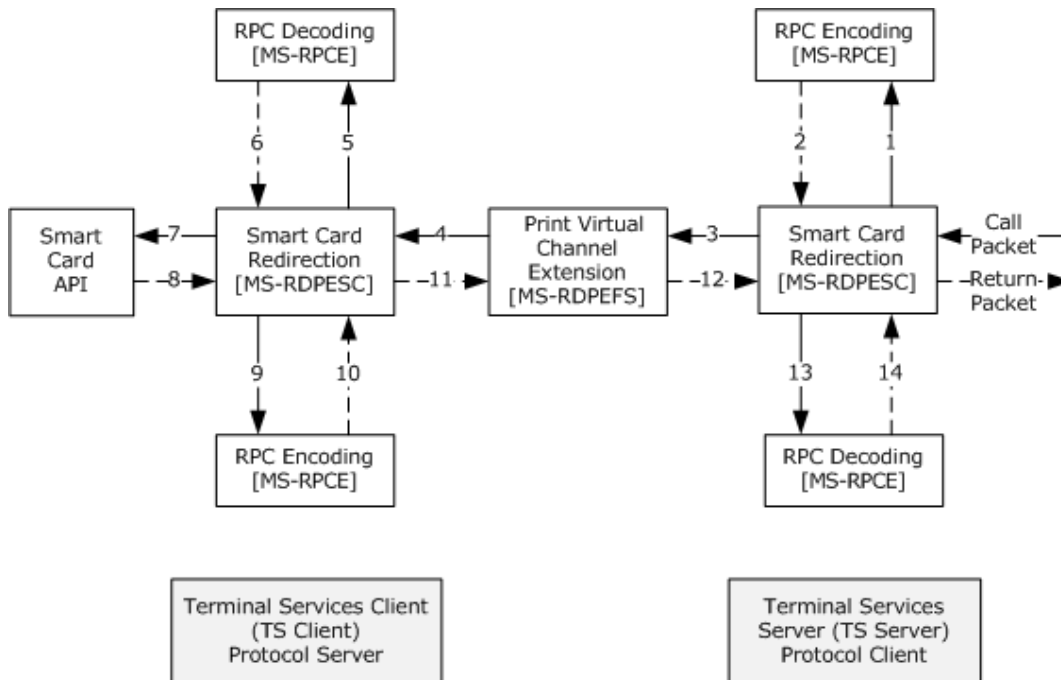
Smart Card Redirection redirects the TS client-side Smart Cards for Windows. When Smart Card Redirection is in effect, TS server application smart card subsystem calls (for example, EstablishContext) are automatically remapped to the TS client-side Smart Cards for Windows, which will then receive the corresponding request. Smart Card Redirection devices are only required to understand one type of device I/O request.

The following figure shows a high-level sequence diagram of the protocol for redirected calls. Device Announce and Device Disconnect are handled via the lower-layer protocols.



**Figure 2: High-level protocol sequence**

The following figure specifies how the messages are encoded and routed from a TS client to a TS server. The following numbered list details corresponding actions related to the pictured protocol flow.



**Figure 3: Protocol flow**

The input for this protocol (call packet) is a combination of an I/O control (IOCTL) and the corresponding structure as specified in section 3.2.5.

1. The call packet structure is encoded as specified in [MS-RPCE] section 2.2.6.
2. The packet, as specified in [MS-RPCE], is returned as a response to 1.
3. The encoded value from 2 is combined with the IOCTL and transported over RDP Device Redirection, as specified in [MS-RDPEFS] section 2.
4. On the TS client, Remote Desktop Protocol: File System Virtual Channel Extension will route the packet from 3 to protocol server for the Smart Card Redirection, as specified in [MS-RDPEFS] section 2.
5. After Smart Card Redirection receives the message, the encoded structure is decoded, as specified in [MS-RPCE] section 2.2.6.
6. The packet, decoded as specified in [MS-RPCE], is a response to 5.
7. Based on the IOCTL, the structure members are used as input parameters to the Smart Cards for Windows, as specified in [PCSC5] section 3.
8. The output parameters including the return code are packaged into the return packet structure for this IOCTL.
9. The return packet structure is encoded as specified in [MS-RPCE] section 2.2.6.
10. Return data, encoded as specified in [MS-RPCE], is a response to 9.
11. The encoded value from 10 is sent to RDP Device Redirection (as specified in [MS-RDPEFS]) as a reply to the call packet from 4.
12. RDP Device Redirection (as specified in [MS-RDPEFS]) routes the reply back to the protocol client.
13. On receipt of packet from 12, the encoded structure is decoded as specified by to [MS-RPCE] section 2.2.6.
14. In response to 13, return data is decoded as specified by [MS-RPCE].

The output from the Smart Card Redirection is the return packet. This data will then be processed by higher layers.

#### **1.4 Relationship to Other Protocols**

This protocol extension expands Remote Desktop Protocol: File System Virtual Channel Extension [MS-RDPEFS] functionality to provide support for Smart Cards for Windows.

This protocol relies on the Distributed Component Object Model (DCOM) Remote Protocol [MS-DCOM], which uses remote procedure call (RPC) as its transport.

This protocol uses the Remote Procedure Call Protocol Extensions ([MS-RPCE] section 2) to encode packet structures carried within an RDP session.

#### **1.5 Prerequisites/Preconditions**

RDP Device Redirection transport (as specified in [MS-RDPEFS] section 2.2.2.7.5) must be configured to redirect smart card devices.

## 1.6 Applicability Statement

This specification applies to redirecting Smart Cards for Windows API-based calls for a Terminal Services client, as specified in [PCSC5] section 3.

## 1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- Protocol Versions: Smart Card Redirection supports the dialects SCREDIR\_VERSION\_XP (1), SCREDIR\_VERSION\_LONGHORN (2), and SCREDIR\_VERSION\_WINDOWS\_8 (3).
- Capability Negotiation: The Smart Card Redirection protocol does not support negotiation of the dialect to use. Instead, an implementation is configured with the dialect to use.

The TS server determines the dialect to use by analyzing the client build number on device announce as specified in [MS-RDPBCGR] section 2.2.1.3.2 using the following mapping. <1>

Build Number	Dialect
>= 7865	SCREDIR_VERSION_WINDOWS_8 (3)
>= 4034 and < 7865	SCREDIR_VERSION_LONGHORN (2)
< 4034	SCREDIR_VERSION_XP (1)

## 1.8 Vendor-Extensible Fields

This protocol uses HRESULTs as defined in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values, provided that they set the C bit (0x20000000) for each vendor-defined value, indicating that the value is a customer code.

This protocol uses Win32 error codes. These values are taken from the Windows error number space, as specified in [MS-ERREF] section 2.2. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

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

IOCTL fields used in this specification are extensible. Vendors MUST implement the corresponding functions.

## 1.9 Standards Assignments

This protocol uses the following RPC UUID for the type\_scard\_pack interface.

Parameter	Value	Reference
Remote procedure call (RPC) interface universally unique identifier (UUID)	A35AF600-9CF4-11CD-A076-08002B2BD711	[C706] Appendix A 2.5





## 2 Messages

The following sections specify how Remote Desktop Protocol: Smart Card Virtual Channel Extension messages are transported, and common data types.

### 2.1 Transport

All messages MUST be transported over established RDP Device Extensions (as specified in [MS-RDPEFS] section 2.1). This protocol uses the device enumerate and announcement messages, as specified in [MS-RDPEFS] section 3.

Remote Desktop Protocol: File System Virtual Channel Extension is responsible for providing a unique Device ID as defined in [MS-RDPEFS] section 3.1.1.

### 2.2 Common Data Types

All structures in this section MUST be encoded as specified in [MS-RPCE] section 2. Unless otherwise stated, the structure MUST be initialized to zero before use.

#### 2.2.1 Common Structures

The structures defined in the following sections are common among both TS server-generated structures (for more information, see section 2.2.2) and TS client-generated structures (for more information, see section 2.2.3).

##### 2.2.1.1 REDIR\_SCARDCONTEXT

REDIR\_SCARDCONTEXT represents a context to Smart Cards for Windows on the TS client.

```
typedef struct _REDIR_SCARDCONTEXT {
    [range(0,16)] unsigned long cbContext;
    [unique] [size_is(cbContext)] byte *pbContext;
} REDIR_SCARDCONTEXT;
```

**cbContext:** The number of bytes in the **pbContext** field.

**pbContext:** An array of **cbContext** bytes that contains Smart Cards for Windows context. The data is implementation-specific and MUST NOT be interpreted or changed on the Protocol server.

##### 2.2.1.2 REDIR\_SCARDHANDLE

REDIR\_SCARDHANDLE represents a smart card reader handle associated with Smart Cards for Windows context.

```
typedef struct _REDIR_SCARDHANDLE {
    REDIR_SCARDCONTEXT Context;
    [range(0,16)] unsigned long cbHandle;
    [size_is(cbHandle)] byte *pbHandle;
} REDIR_SCARDHANDLE;
```

**Context:** A valid context, as specified in REDIR\_SCARDCONTEXT.

**cbHandle:** The number of bytes in the **pbHandle** field.

**pbHandle:** An array of **cbHandle** bytes that corresponds to a smart card reader handle on the TS client. The data is implementation-specific and MUST NOT be interpreted or changed on the Protocol server.

### 2.2.1.3 Connect\_Common

The Connect\_Common structure contains information common to both versions of the Connect function (for more information, see sections 2.2.2.13 and 2.2.2.14).

```
typedef struct _Connect_Common {
    REDIR_SCARDCONTEXT Context;
    unsigned long dwShareMode;
    unsigned long dwPreferredProtocols;
} Connect_Common;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**dwShareMode:** A flag that indicates whether other applications are allowed to form connections to the card. Possible values of this field are specified in section 2.2.6.

**dwPreferredProtocols:** A bitmask of acceptable protocols for the connection, as specified in section 2.2.5.

### 2.2.1.4 LocateCards\_ATRMask

The LocateCards\_ATRMask structure contains the information to identify a card type.

```
typedef struct _LocateCards_ATRMask {
    [range(0,36)] unsigned long cbAtr;
    byte rgbAtr[36];
    byte rgbMask[36];
} LocateCards_ATRMask;
```

**cbAtr:** The number of bytes used in the **rgbAtr** and **rgbMask** fields.

**rgbAtr:** Values for the card's Answer To Reset (ATR) string. This value MUST be formatted as specified in [ISO/IEC-7816-3] section 8. Unused bytes MUST be set to 0 and MUST be ignored.

**rgbMask:** Values for the mask for the card's ATR string. Each bit that cannot vary between cards of the same type MUST be set to 1. Unused bytes MUST be set to 0 and MUST be ignored.

### 2.2.1.5 ReaderState\_Common\_Call

The ReaderState\_Common\_Call structure contains the state of the reader at the time of the call as seen by the caller.

```
typedef struct _ReaderState_Common_Call {
    unsigned long dwCurrentState;
    unsigned long dwEventState;
    [range(0,36)] unsigned long cbAtr;
    byte rgbAtr[36];
} ReaderState_Common_Call;
```

**dwCurrentState:** A bitmap that specifies the current reader state according to the TS client. Possible values are specified in section 2.2.7.

**dwEventState:** A bitmap that defines the state of the reader after a state change. Possible values are specified in section 2.2.7.

**cbAtr:** The number of bytes used in the ATR string.

**rgbAtr:** The value for the card's ATR string. If **cbAtr** is NOT zero, this value MUST be formatted in accordance to [ISO/IEC-7816-3] section 8. Unused bytes MUST be set to 0 and MUST be ignored.

### 2.2.1.6 ReaderStateA

The ReaderStateA structure contains information used in calls that only require Smart Cards for Windows context and an ASCII string.

```
typedef struct _ReaderStateA {
    [string] const char* szReader;
    ReaderState_Common_Call Common;
} ReaderStateA;
```

**szReader:** An ASCII string specifying the reader name.

**Common:** A packet that specifies the state of the reader at the time of the call. For information about this packet, see section 2.2.1.5.

### 2.2.1.7 ReaderStateW

The ReaderStateW structure is a Unicode representation of the state of a smart card reader.

```
typedef struct _ReaderStateW {
    [string] const wchar_t* szReader;
    ReaderState_Common_Call Common;
} ReaderStateW;
```

**szReader:** A Unicode string specifying the reader name.

**Common:** A packet that specifies the state of the reader at the time of the call. For information about this packet, see section 2.2.1.5.

### 2.2.1.8 SCardIO\_Request

The SCardIO\_Request structure represents the data to be prepended to a Transmit command (for more information, see section 3.1.4.35).

```
typedef struct _SCardIO_Request {
    unsigned long dwProtocol;
    [range(0,1024)] unsigned long cbExtraBytes;
    [unique] [size_is(cbExtraBytes)] byte *pbExtraBytes;
} SCardIO_Request;
```

**dwProtocol:** The protocol in use. Possible values are specified in section 2.2.5.

**cbExtraBytes:** The number of bytes in the **pbExtraBytes** field.

**pbExtraBytes:** Request data.

### 2.2.1.9 ReadCache\_Common

The ReadCache\_Common structure contains information common to both the ReadCacheA\_Call and ReadCacheW\_Call structures.

```
typedef struct _ReadCache_Common {
    REDIR_SCARDCONTEXT Context;
    UUID* CardIdentifier;
    unsigned long FreshnessCounter;
    long fPbDataIsNull;
    unsigned long cbDataLen;
} ReadCache_Common;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**CardIdentifier:** A UUID that specifies the name of the smart card with which the name-value pair is associated.

**FreshnessCounter:** A value specifying the current revision of the data.

**fPbDataIsNull:** A Boolean value specifying whether the caller wants to retrieve the length of the data. It MUST be set to TRUE (0x00000001) if the caller wants only to retrieve the length of the data; otherwise, it MUST be set to FALSE (0x00000000).

**cbDataLen:** The length of the buffer specified on the server side. If **cbDataLen** is set to SCARD\_AUTOALLOCATE with a value of 0xFFFFFFFF, a buffer of any length can be returned. Otherwise, the returned buffer MUST NOT exceed **cbDataLen** bytes. This field MUST be ignored if **fPbDataIsNull** is set to TRUE (0x00000001).

### 2.2.1.10 WriteCache\_Common

The WriteCache\_Common structure contains information common between the WriteCacheA\_Call and WriteCacheW\_Call structures.

```
typedef struct _WriteCache_Common {
    REDIR_SCARDCONTEXT Context;
    UUID *CardIdentifier;
    unsigned long FreshnessCounter;
    [range(0,65536)] unsigned long cbDataLen;
    [unique] [size_is(cbDataLen)] byte *pbData;
} WriteCache_Common;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**CardIdentifier:** A UUID that identifies the smart card with which the data SHOULD be stored. CardIdentifier MUST be a unique value per the smart card.

**FreshnessCounter:** A value specifying the current revision of the data.

**cbDataLen:** The number of bytes in the **pbData** field.

**pbData:** **cbDataLen** bytes of data to be stored.

### 2.2.1.11 ReaderState\_Return

The ReaderState\_Return structure specifies state information returned from Smart Cards for Windows.

```
typedef struct _ReaderState_Return {
```

```

    unsigned long dwCurrentState;
    unsigned long dwEventState;
    [range(0,36)] unsigned long cbAtr;
    byte rgbAtr[36];
} ReaderState_Return;

```

**dwCurrentState:** A bitmap that defines the current state of the reader at the time of the call. Possible values are specified in section 2.2.7.

**dwEventState:** A bitmap that defines the state of the reader after a state change as seen by Smart Cards for Windows. Possible values are specified in section 2.2.7.

**cbAtr:** The number of used bytes in **rgbAtr**.

**rgbAtr:** The values for the card's ATR string. Unused bytes MUST be set to zero and MUST be ignored on receipt.

## 2.2.2 TS Server-Generated Structures

All structures in this section are sent from the TS server to the TS client.

### 2.2.2.1 EstablishContext\_Call

The EstablishContext\_Call structure is used to specify the scope of Smart Cards for Windows context to be created (for more information, see section 3.1.4.1).

```

typedef struct _EstablishContext_Call {
    unsigned long dwScope;
} EstablishContext_Call;

```

**dwScope:** The scope of the context that will be established. The following table shows valid values of this field.

Value	Meaning
SCARD_SCOPE_USER 0x00000000	The context is a user context; any database operations MUST be performed with the domain of the user.
SCARD_SCOPE_TERMINAL 0x00000001	The context is a terminal context; any database operations MUST be performed with the domain of the terminal. This flag is currently unused; it is here for compatibility with [PCSC5] section 3.1.3.
SCARD_SCOPE_SYSTEM 0x00000002	The context is the system context; any database operations MUST be performed within the domain of the system.

### 2.2.2.2 Context\_Call

The Context\_Call structure contains Smart Cards for Windows context.

```

typedef struct _Context_Call {
    REDIR_SCARDCONTEXT Context;
} Context_Call;

```

**Context:** A valid context, as specified in section 2.2.1.1.

### 2.2.2.3 ListReaderGroups\_Call

The ListReaderGroups\_Call structure contains the parameters for the List Readers Groups call (for more information, see sections 3.1.4.5 and 3.1.4.6).

```
typedef struct _ListReaderGroups_Call {
    REDIR_SCARDCONTEXT Context;
    long fmszGroupsIsNull;
    unsigned long cchGroups;
} ListReaderGroups_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**fmszGroupsIsNull:** A Boolean value specifying whether the caller wants to retrieve just the length of the data. Set to FALSE (0x00000000) in order to allow the data to be returned. Set to TRUE (0x00000001) and only the length of the data will be returned.

**cchGroups:** The length of the string buffer specified by the caller. If **cchGroups** is set to SCARD\_AUTOALLOCATE with a value of 0xFFFFFFFF, a string of any length can be returned. Otherwise, the returned string MUST NOT exceed **cchGroups** characters in length, including any null characters. When the string to be returned exceeds **cchGroups** characters in length, including any null characters, ListReaderGroups\_Return.ReturnCode MUST be set to SCARD\_E\_INSUFFICIENT\_BUFFER (0x80100008). The **cchGroups** field MUST be ignored if **fmszGroupsIsNull** is set to TRUE (0x00000001). Also, if **fmszGroupsIsNull** is set to FALSE (0x00000000) but **cchGroups** is set to 0x00000000, then the call MUST succeed, ListReaderGroups\_Return.cBytes MUST be set to the length of the data, in bytes, and ListReaderGroups\_Return.msiz MUST be set to NULL.

### 2.2.2.4 ListReaders\_Call

The ListReaders\_Call structure contains the parameters for the List Readers call (for more information, see sections 3.1.4.7 and 3.1.4.8).

```
typedef struct _ListReaders_Call {
    REDIR_SCARDCONTEXT Context;
    [range(0, 65536)] unsigned long cBytes;
    [unique] [size_is(cBytes)] const byte *mszGroups;
    long fmszReadersIsNull;
    unsigned long cchReaders;
} ListReaders_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**cBytes:** The length, in bytes, of reader groups specified in **mszGroups**.

**mszGroups:** The names of the reader groups defined in the system. Reader groups not present on the protocol server MUST be ignored. The value of this is dependent on the context (IOCTL) that it is used.

Value	Meaning
SCARD_IOCTL_LISTREADERSA 0x00090028	ASCII multistring
SCARD_IOCTL_LISTREADERSW 0x0009002C	Unicode multistring

**fmszReadersIsNull:** A Boolean value specifying whether the caller wants to retrieve the length of the data. Set to FALSE (0x00000000) to allow the data to be returned. Set to TRUE (0x00000001), and only the length of the data will be returned.

**cchReaders:** The length of the string buffer specified by the caller. If **cchReaders** is set to SCARD\_AUTOALLOCATE with a value of 0xFFFFFFFF, a string of any length can be returned. Otherwise, the returned string MUST NOT exceed **cchReaders** characters in length, including any NULL characters. When the string to be returned exceeds **cchReaders** characters in length, including any null characters, ListReaders\_Return.**ReturnCode** MUST be set to SCARD\_E\_INSUFFICIENT\_BUFFER (0x80100008). The **cchReaders** field MUST be ignored if **fmszReadersIsNull** is set to TRUE (0x00000001). Also, if **fmszReadersIsNull** is set to FALSE (0x00000000) but **cchReaders** is set to 0x00000000, then the call MUST succeed, ListReaders\_Return.**cBytes** MUST be set to the length of the data in bytes, and ListReaders\_Return.**msz** MUST be set to NULL.

### 2.2.2.5 ContextAndStringA\_Call

The ContextAndStringA\_Call structure contains information used in calls that only require a Smart Cards for Windows context and an ASCII string.

```
typedef struct _ContextAndStringA_Call {  
    REDIR_SCARDCONTEXT Context;  
    [string] const char* sz;  
} ContextAndStringA_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**sz:** The value of this string depends on the context (based on IOCTL) in which this structure is used.

Value	Meaning
SCARD_IOCTL_INTRODUCEREADERGROUPA 0x00090050	Reader group name
SCARD_IOCTL_FORGETREADERGROUPA 0x00090058	Reader group name
SCARD_IOCTL_FORGETREADERA 0x00090068	Reader name

### 2.2.2.6 ContextAndStringW\_Call

The ContextAndStringW\_Call structure contains information used in calls that only require a Smart Cards for Windows context and a Unicode string.

```
typedef struct _ContextAndStringW_Call {  
    REDIR_SCARDCONTEXT Context;  
    [string] const wchar_t* sz;  
} ContextAndStringW_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**sz:** The value of this Unicode string depends on the context (based on IOCTL) in which this structure is used.



Value	Meaning
SCARD_IOCTL_INTRODUCEREADERGROUPW 0x00090054	Reader group name
SCARD_IOCTL_FORGETREADERGROUPW 0x0009005C	Reader group name
SCARD_IOCTL_FORGETREADERW 0x0009006C	Reader name

### 2.2.2.7 ContextAndTwoStringA\_Call

The contents of the ContextAndTwoStringA\_Call structure are used in those calls that require a valid Smart Cards for Windows context (as specified in section 3.2.5) and two strings (friendly names).

```
typedef struct _ContextAndTwoStringA_Call {
    REDIR_SCARDCONTEXT Context;
    [string] const char* sz1;
    [string] const char* sz2;
} ContextAndTwoStringA_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**sz1:** The value of this ASCII string depends on the context (based on IOCTL) in which it is used.

Value	Meaning
SCARD_IOCTL_INTRODUCEREADERA 0x00090060	Reader name
SCARD_IOCTL_ADDREADERTOGROUPA 0x00090070	Reader name
SCARD_IOCTL_REMOVEREADERFROMGROUPA 0x00090078	Reader name

**sz2:** The value of this ASCII string depends on the context (based on IOCTL) in which it is used.

Value	Meaning
SCARD_IOCTL_INTRODUCEREADERA 0x00090060	Device name
SCARD_IOCTL_ADDREADERTOGROUPA 0x00090070	Reader group name
SCARD_IOCTL_REMOVEREADERFROMGROUPA 0x00090078	Reader group name

### 2.2.2.8 ContextAndTwoStringW\_Call

The contents of the ContextAndTwoStringW\_Call structure is used in those calls that require a valid Smart Cards for Windows context (as specified in section 3.2.5) and two strings (friendly names).

```
typedef struct _ContextAndTwoStringW_Call {
    REDIR_SCARDCONTEXT Context;
    [string] const wchar_t* sz1;
    [string] const wchar_t* sz2;
} ContextAndTwoStringW_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**sz1:** The value of this Unicode string depends on the context (based on IOCTL) in which it is used.

Value	Meaning
SCARD_IOCTL_INTRODUCEREADERW 0x00090064	Reader name
SCARD_IOCTL_ADDREADERTOGROUPW 0x00090074	Reader name
SCARD_IOCTL_REMOVEREADERFROMGROUPW 0x0009007C	Reader name

**sz2:** The value of this Unicode string depends on the context (based on IOCTL) in which it is used.

Value	Meaning
SCARD_IOCTL_INTRODUCEREADERW 0x00090064	Device name
SCARD_IOCTL_ADDREADERTOGROUPW 0x00090074	Reader group name
SCARD_IOCTL_REMOVEREADERFROMGROUPW 0x0009007C	Reader group name

### 2.2.2.9 LocateCardsA\_Call

The parameters of the LocateCardsA\_Call structure specify the list of smart card readers to search for the specified card types. For call information, see section 3.1.4.21.

```
typedef struct _LocateCardsA_Call {
    REDIR_SCARDCONTEXT Context;
    [range(0, 65536)] unsigned long cBytes;
    [size_is(cBytes)] const byte* mszCards;
    [range(0,10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateA* rgReaderStates;
} LocateCardsA_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**cBytes:** The number of bytes in the **mszCards** field.

**mszCards:** An ASCII multistring of card names to locate. Card names MUST be registered in Smart Cards for Windows. Unknown card types MUST be ignored.

**cReaders:** The number of reader state structures.

**rgReaderStates:** The reader state information specifying which readers are searched for the cards listed in **mszCards**.

### 2.2.2.10 LocateCardsW\_Call

The parameters of the LocateCardsW\_Call structure specify the list of smart card readers to search for the specified card types. For more information, see section 3.1.4.22.

```
typedef struct _LocateCardsW_Call {
    REDIR_SCARDCONTEXT Context;
    [range(0,65536)] unsigned long cBytes;
    [size_is(cBytes)] const byte* mszCards;
    [range(0,10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateW* rgReaderStates;
} LocateCardsW_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**cBytes:** The number of bytes in the **mszCards** field.

**mszCards:** A Unicode multistring of card names to locate. Card names MUST be registered in Smart Cards for Windows. Unknown card types MUST be ignored.

**cReaders:** The number of reader state structures.

**rgReaderStates:** The reader state information used to locate the cards listed in *mszCards*.

### 2.2.2.11 GetStatusChangeA\_Call

The GetStatusChangeA\_Call structure provides the state change in the reader as specified in section 3.1.4.23.

```
typedef struct _GetStatusChangeA_Call {
    REDIR_SCARDCONTEXT Context;
    unsigned long dwTimeOut;
    [range(0,11)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateA* rgReaderStates;
} GetStatusChangeA_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**dwTimeOut:** The maximum amount of time, in milliseconds, to wait for an action. If this member is set to 0xFFFFFFFF (INFINITE), the caller MUST wait until an action occurs.

**cReaders:** The number of ReaderStates to track.

**rgReaderStates:** Smart card readers that the caller is tracking.

### 2.2.2.12 GetStatusChangeW\_Call

The GetStatusChangeW\_Call structure provides the state change in the Reader as specified in section 3.1.4.24.

```

typedef struct _GetStatusChangeW_Call {
    REDIR_SCARDCONTEXT Context;
    unsigned long dwTimeOut;
    [range(0,11)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateW* rgReaderStates;
} GetStatusChangeW_Call;

```

**Context:** A valid context, as specified in section 2.2.1.1.

**dwTimeOut:** Maximum amount of time, in milliseconds, to wait for an action. If set to 0xFFFFFFFF (INFINITE), the caller MUST wait until an action occurs.

**cReaders:** The number of ReaderStates to track.

**rgReaderStates:** Smart card readers that the caller is tracking.

### 2.2.2.13 ConnectA\_Call

ConnectA\_Call opens a connection to the smart card located in the reader identified by a reader name.

```

typedef struct _ConnectA_Call {
    [string] const char* szReader;
    Connect_Common Common;
} ConnectA_Call;

```

**szReader:** An ASCII string specifying the reader name to connect to.

**Common:** Additional parameters that are required for the Connect call are specified in section 3.1.4.28. For more information, see section 2.2.1.3.

### 2.2.2.14 ConnectW\_Call

The ConnectW\_Call structure is used to open a connection to the smart card located in the reader identified by a reader name.

```

typedef struct _ConnectW_Call {
    [string] const wchar_t* szReader;
    Connect_Common Common;
} ConnectW_Call;

```

**szReader:** A Unicode string specifying the reader name to connect to.

**Common:** Additional parameters that are required for the Connect call. For more information, see sections 3.1.4.29 and 2.2.1.3.

### 2.2.2.15 Reconnect\_Call

The Reconnect\_Call structure is used to reopen a connection to the smart card associated with a valid context. For more information, see section 3.1.4.36.

```

typedef struct _Reconnect_Call {
    REDIR_SCARDHANDLE hCard;
    unsigned long dwShareMode;
    unsigned long dwPreferredProtocols;
    unsigned long dwInitialization;
} Reconnect_Call;

```

**hCard:** A handle, as specified in section 2.2.1.2.

**dwShareMode:** A flag that indicates whether other applications can form connections to this card. For acceptable values of this field, see section 2.2.6.

**dwPreferredProtocols:** A bit mask of acceptable protocols for this connection. For specifics on possible values, see section 2.2.5.

**dwInitialization:** A type of initialization that SHOULD be performed on the card.

Value	Meaning
SCARD_LEAVE_CARD 0x00000000	Do not do anything.
SCARD_RESET_CARD 0x00000001	Reset the smart card.
SCARD_UNPOWER_CARD 0x00000002	Turn off and reset the smart card.

### 2.2.2.16 HCardAndDisposition\_Call

The HCardAndDisposition\_Call structure defines the action taken on the disposition of a smart card associated with a valid context when a connection is terminated.

```
typedef struct _HCardAndDisposition_Call {  
    REDIR_SCARDHANDLE hCard;  
    unsigned long dwDisposition;  
} HCardAndDisposition_Call;
```

**hCard:** A handle, as specified in section 2.2.1.2.

**dwDisposition:** The action to take on the card in the connected reader upon close. This value is ignored on a BeginTransaction message call, as specified in section 3.2.5.3.61.

Value	Meaning
SCARD_LEAVE_CARD 0x00000000	Do not do anything.
SCARD_RESET_CARD 0x00000001	Reset the smart card.
SCARD_UNPOWER_CARD 0x00000002	Turn off and reset the smart card.
SCARD_EJECT_CARD 0x00000003	Eject the smart card.

### 2.2.2.17 State\_Call

The State\_Call structure defines parameters to the State call (as specified in section 3.1.4.40) for querying the contents of a smart card reader.

```
typedef struct _State_Call {
    REDIR_SCARDHANDLE hCard;
    long fpbAttrIsNULL;
    unsigned long cbAttrLen;
} State_Call;
```

**hCard:** A handle, as specified in section 2.2.1.2.

**fpbAttrIsNULL:** A Boolean value specifying whether the caller wants to retrieve the length of the data. Set to FALSE (0x00000000) to allow the data to be returned. Set to TRUE (0x00000001), and only the length of the data will be returned. SHOULD be set to TRUE if **cbAttrLen** is set to SCARD\_AUTOALLOCATE (0xFFFFFFFF).

Name	Value
FALSE	0x00000000
TRUE	0x00000001

**cbAttrLen:** The length of the buffer specified on the TS server side. If **cbAttrLen** is set to SCARD\_AUTOALLOCATE with a value of 0xFFFFFFFF, an array of any length can be returned. Otherwise, the returned array MUST NOT exceed **cbAttrLen** bytes in length. When the array to be returned exceeds **cbAttrLen** bytes in length, State\_Return.ReturnCode MUST be set to SCARD\_E\_INSUFFICIENT\_BUFFER (0x80100008). Also, **cbAttrLen** is ignored if **fpbAttrIsNULL** is set to TRUE (0x00000001). If **fpbAttrIsNULL** is set to FALSE (0x00000000) but **cbAttrLen** is set to 0x00000000, then the call MUST succeed, State\_Return.cbAttrLen MUST be set to the length of the data in bytes, and State\_Return.rgAttr MUST be set to NULL.

### 2.2.2.18 Status\_Call

Status\_Call obtains the status of a connection for a valid smart card reader handle.

```
typedef struct _Status_Call {
    REDIR_SCARDHANDLE hCard;
    long fmszReaderNamesIsNULL;
    unsigned long cchReaderLen;
    unsigned long cbAttrLen;
} Status_Call;
```

**hCard:** A handle, as specified in section 2.2.1.2.

**fmszReaderNamesIsNULL:** A Boolean value specifying whether the caller wants to retrieve the length of the data. Set to FALSE (0x00000000) to allow the data to be returned. Set to TRUE (0x00000001), and only the length of the data will be returned. Also, **cchReaderLen** is ignored if this value is TRUE (0x00000001).

Name	Value
FALSE	0x00000000
TRUE	0x00000001

**cchReaderLen:** The length of the string buffer specified on the TS server side. If **cchReaderLen** is set to **SCARD\_AUTOALLOCATE** with a value of **0xFFFFFFFF**, a string of any length can be returned. Otherwise, the returned string **MUST NOT** exceed **cchReaderLen** characters in length, including any null characters. When the string to be returned exceeds **cchReaderLen** characters in length, including any null characters, **Status\_Return.ReturnCode** **MUST** be set to **SCARD\_E\_INSUFFICIENT\_BUFFER** (**0x80100008**). The **cchReaderLen** field **MUST** be ignored if **fmszReaderNamesIsNULL** is **TRUE** (**0x00000001**). Also, if **fmszReaderNamesIsNULL** is set to **FALSE** (**0x00000000**) but **cchReaderLen** is set to **0x00000000**, then the call **MUST** succeed, **Status\_Return.cbAtrLen** **MUST** be set to the length of the data in bytes, and **Status\_Return.pbAtr** **MUST** be set to **NULL**.

**cbAtrLen:** Unused. **MUST** be ignored upon receipt.

### 2.2.2.19 Transmit\_Call

The **Transmit\_Call** structure is used to send data to the smart card associated with a valid context.

```
typedef struct _Transmit_Call {
    REDIR_SCARDHANDLE hCard;
    SCardIO_Request ioSendPci;
    [range(0,66560)] unsigned long cbSendLength;
    [size_is(cbSendLength)] const byte* pbSendBuffer;
    [unique] SCardIO_Request* pioRecvPci;
    long fpbRecvBufferIsNULL;
    unsigned long cbRecvLength;
} Transmit_Call;
```

**hCard:** A handle, as specified in section 2.2.1.2.

**ioSendPci:** A packet specifying input header information as specified in section 2.2.1.8.

**cbSendLength:** The length, in bytes, of the **pbSendBuffer** field.

**pbSendBuffer:** The data to be written to the card. The format of the data is specific to an individual card. For more information about data formats, see [ISO/IEC-7816-4] sections 5 through 7.

**pioRecvPci:** If non-NULL, this field is an **SCardIO\_Request** packet that is set up in the same way as the **ioSendPci** field and passed as the **pioRecvPci** parameter of the **Transmit** call. If the value of this is **NULL**, the caller is not requesting the **pioRecvPci** value to be returned.

**fpbRecvBufferIsNULL:** A Boolean value specifying whether the caller wants to retrieve the length of the data. **MUST** be set to **TRUE** (**0x00000001**) if the caller wants only to retrieve the length of the data; otherwise, it **MUST** be set to **FALSE** (**0x00000000**).

Name	Value
FALSE	0x00000000
TRUE	0x00000001

**cbRecvLength:** The maximum size of the buffer to be returned. **MUST** be ignored if **fpbRecvBufferIsNULL** is set to **TRUE** (**0x00000001**).

### 2.2.2.20 Control\_Call

Normally, communication is to the smart card via the reader. However, in some cases, the ability to communicate directly with the smart card reader is requested. The **Control\_Call** structure provides the ability to talk to the reader.

```

typedef struct _Control_Call {
    REDIR_SCARDHANDLE hCard;
    unsigned long dwControlCode;
    [range(0,66560)] unsigned long cbInBufferSize;
    [unique] [size_is(cbInBufferSize)] const byte *pvInBuffer;
    long fpvOutBufferIsNULL;
    unsigned long cbOutBufferSize;
} Control_Call;

```

**hCard:** A handle, as specified in section 2.2.1.2.

**dwControlCode:** The control code for the operation. These values are specific to the hardware device. This protocol MUST NOT restrict or define any values for this control codes.

**cbInBufferSize:** The size in bytes of the **pvInBuffer** field.

**pvInBuffer:** A buffer that contains the data required to perform the operation. This field SHOULD be NULL if the **dwControlCode** field specifies an operation that does not require input data. Otherwise, this data is specific to the function being performed.

**fpvOutBufferIsNULL:** A Boolean value specifying whether the caller wants to retrieve the length of the data. MUST be set to TRUE (0x00000001) if the caller wants only to retrieve the length of the data; otherwise, it MUST be set to FALSE (0x00000000).

Name	Value
FALSE	0x00000000
TRUE	0x00000001

**cbOutBufferSize:** The maximum size of the buffer to be returned. This field MUST be ignored if fpvOutBufferIsNULL is set to TRUE (0x00000001).

### 2.2.2.21 GetAttrib\_Call

The GetAttrib\_Call structure is used to read smart card reader attributes.

```

typedef struct _GetAttrib_Call {
    REDIR_SCARDHANDLE hCard;
    unsigned long dwAttrId;
    long fpbAttrIsNULL;
    unsigned long cbAttrLen;
} GetAttrib_Call;

```

**hCard:** A handle, as specified in section 2.2.1.2.

**dwAttrId:** An identifier for the attribute to get. For more information on defined attributes, see [PCSC3] section 3.1.2.

**fpbAttrIsNULL:** A Boolean value specifying whether the caller wants to retrieve the length of the data. Set to FALSE (0x00000000) in order to allow the data to be returned. Set to TRUE (0x00000001) and only the length of the data will be returned.

Name	Value
FALSE	0x00000000
TRUE	0x00000001



**cbAttrLen:** The length of the buffer specified on the TS Server side. If **cbAttrLen** is set to SCARD\_AUTOALLOCATE with a value of 0xFFFFFFFF then any buffer length can be returned. Otherwise, the returned buffer MUST NOT exceed **cbAttrLen** bytes in length. When the buffer to be returned exceeds **cbAttrLen** bytes in length, GetAttrib\_Return.**ReturnCode** MUST be set to SCARD\_E\_INSUFFICIENT\_BUFFER (0x80100008). The **cbAttrLen** field MUST be ignored if **fpbAttrIsNull** is set to TRUE (0x00000001). Also, if **fpbAttrIsNull** is set to FALSE (0x00000000) but **cbAttrLen** is set to 0x00000000, then the call MUST succeed, GetAttrib\_Return.**cbAttrLen** MUST be set to the length of the data, in bytes, and GetAttrib\_Return.**pbAttr** MUST be set to NULL.

### 2.2.2.22 SetAttrib\_Call

The SetAttrib\_Call structure allows users to set smart card reader attributes.

```
typedef struct _SetAttrib_Call {
    REDIR_SCARDHANDLE hCard;
    unsigned long dwAttrId;
    [range(0,65536)] unsigned long cbAttrLen;
    [size_is(cbAttrLen)] const byte* pbAttr;
} SetAttrib_Call;
```

**hCard:** A handle, as specified in section 2.2.1.2.

**dwAttrId:** The identifier of the attribute to set. The values are write-only. For more information on possible values, see [PCSC3] section 3.1.2.

**cbAttrLen:** The size, in bytes, of the data corresponding to the **pbAttr** field.

**pbAttr:** A buffer that contains the attribute whose identifier is supplied in the **dwAttrId** field. The format is specific to the value being set.

### 2.2.2.23 LocateCardsByATRA\_Call

The LocateCardsByATRA\_Call structure returns information concerning the status of the smart card of interest (ATR).

```
typedef struct _LocateCardsByATRA_Call {
    REDIR_SCARDCONTEXT Context;
    [range(0,1000)] unsigned long cAtrs;
    [size_is(cAtrs)] LocateCards_ATRMask* rgAttrMasks;
    [range(0,10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateA* rgReaderStates;
} LocateCardsByATRA_Call;
```

**Context:** A valid context, as specified in section 2.2.2.13.

**cAtrs:** The number of bytes in the **rgAttrMasks** field.

**rgAttrMasks:** An array of ATRs to match against currently inserted cards.

**cReaders:** The number of elements in the **rgReaderStates** field.

**rgReaderStates:** The states of the readers that the application is monitoring. The states reflect what the application determines to be the current states of the readers and that might differ from the actual states.

### 2.2.2.24 LocateCardsByATRW\_Call

The LocateCardsByATRW\_Call structure returns information concerning the status of the smart card of interest (ATR).

```
typedef struct _LocateCardsByATRW_Call {
    REDIR_SCARDCONTEXT Context;
    [range(0, 1000)] unsigned long cAtrs;
    [size_is(cAtrs)] LocateCards_ATRMask* rgAtrMasks;
    [range(0,10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateW* rgReaderStates;
} LocateCardsByATRW_Call;
```

**Context:** A valid context, as specified in section 2.2.2.14.

**cAtrs:** The number of bytes in the **rgAtrMasks** field.

**rgAtrMasks:** An array of ATRs to match against currently inserted cards.

**cReaders:** The number of elements in the **rgReaderStates** field.

**rgReaderStates:** The states of the readers that the application is monitoring. The states reflects what the application believes is the current states of the readers and might differ from the actual states.

### 2.2.2.25 ReadCacheA\_Call

The ReadCacheA\_Call structure is used to obtain the card and reader information from the cache.

```
typedef struct _ReadCacheA_Call {
    [string] char* szLookupName;
    ReadCache_Common Common;
} ReadCacheA_Call;
```

**szLookupName:** An ASCII string containing the lookup name.

**Common:** Additional parameters for the Read Cache call (for additional information, see section 3.1.4.42), as specified in section 2.2.1.9.

### 2.2.2.26 ReadCacheW\_Call

The ReadCacheW\_Call structure is used to obtain the card and reader information from the cache.

```
typedef struct _ReadCacheW_Call {
    [string] wchar_t* szLookupName;
    ReadCache_Common Common;
} ReadCacheW_Call;
```

**szLookupName:** A Unicode string containing the lookup name.

**Common:** Additional parameters for the Read Cache call (for additional information, see section 3.1.4.43), as specified in section 2.2.1.9.

### 2.2.2.27 WriteCacheA\_Call

The WriteCacheA\_Call structure is used to write the card and reader information to the cache.

```
typedef struct _WriteCacheA_Call {
    [string] char* szLookupName;
    WriteCache_Common Common;
} WriteCacheA_Call;
```

**szLookupName:** An ASCII string containing the lookup name.

**Common:** Additional parameters for the Write Cache call (for more information, see section 3.1.4.44), as specified in section 2.2.1.10.

### 2.2.2.28 WriteCacheW\_Call

The WriteCacheW\_Call structure is used to write the card and reader information to the cache.

```
typedef struct _WriteCacheW_Call {
    [string] wchar_t* szLookupName;
    WriteCache_Common Common;
} WriteCacheW_Call;
```

**szLookupName:** An Unicode string containing the lookup name.

**Common:** Additional parameters for the Write Cache call (for more information, see section 2.2.1.10).

### 2.2.2.29 GetTransmitCount\_Call

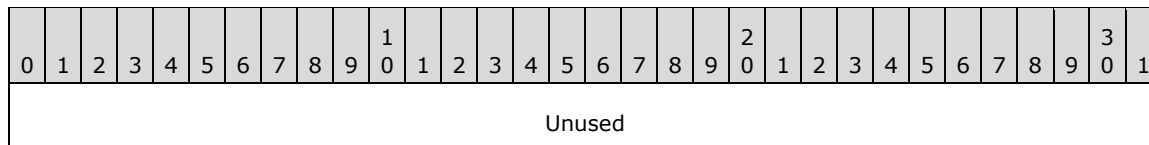
The GetTransmitCount\_Call structure is used to obtain the number of transmit calls sent to the card since the reader was introduced.

```
typedef struct _GetTransmitCount_Call {
    REDIR_SCARDHANDLE hCard;
} GetTransmitCount_Call;
```

**hCard:** A handle, as specified in section 2.2.1.2.

### 2.2.2.30 ScardAccessStartedEvent\_Call

ScardAccessStartedEvent\_Call is just an uninitialized 4-byte buffer that is sent as the IOCTL requires a payload. There is no corresponding serialized structure for this call.



**Unused (4 bytes):** The field is uninitialized. It SHOULD contain random data and MUST be ignored on receipt.

### 2.2.2.31 GetReaderIcon\_Call

The GetReaderIcon\_Call structure is used to obtain the reader icon from the smart card reader's INF file.

```
typedef struct _GetReaderIcon_Call {
    REDIR_SCARDCONTEXT Context;
```

```
[string] wchar_t* szReaderName;  
} GetReaderIcon_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**szReaderName:** A Unicode string containing the reader name.

### 2.2.2.32 GetDeviceTypeId\_Call

The GetDeviceTypeId\_Call structure is used to obtain the reader's device ID from the smart card reader's INF file.

```
typedef struct _GetDeviceTypeId_Call {  
    REDIR_SCARDCONTEXT Context;  
    [string] wchar_t* szReaderName;  
} GetDeviceTypeId_Call;
```

**Context:** A valid context, as specified in section 2.2.1.1.

**szReaderName:** A Unicode string containing the lookup name.

## 2.2.3 TS Client-Generated Structures

These structures originate from the client process and compose part of the return packet. If the **ReturnCode** field of the structure is nonzero, all other fields **MUST** be set to zero and **MUST** be ignored on receipt.

### 2.2.3.1 ReadCache\_Return

The ReadCache\_Return structure is used to obtain the data that corresponds to the lookup item requested in ReadCacheA\_Call as specified in section 2.2.2.25, or ReadCacheW\_Call as specified in section 2.2.2.26. For more call information, see sections 3.1.4.42 and 3.1.4.43.

```
typedef struct _ReadCache_Return {  
    long ReturnCode;  
    [range(0,65536)] unsigned long cbDataLen;  
    [unique] [size_is(cbDataLen)] byte *pbData;  
} ReadCache_Return;
```

**ReturnCode:** HRESULT or Win32 Error codes. Zero indicates success; any other value indicates failure.

**cbDataLen:** The number of bytes in the **pbData** field.

**pbData:** The value of the look up item.

### 2.2.3.2 EstablishContext\_Return

The EstablishContext\_Return structure is used to provide a response to an Establish Context call (for more information, see section 3.1.4.1.)

```
typedef struct _EstablishContext_Return {  
    long ReturnCode;  
    REDIR_SCARDCONTEXT Context;  
} EstablishContext_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**Context:** A valid context, as specified in section 2.2.1.1.

### 2.2.3.3 Long\_Return

The Long\_Return structure is used for return codes for calls that return only a long value.

```
typedef struct _long_Return {  
    long ReturnCode;  
} long_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

### 2.2.3.4 ListReaderGroups\_Return and ListReaders\_Return

The ListReaderGroups\_Return and ListReaders\_Return structures are used to obtain results for those calls that return a multistring, in addition to a long return value. For more information, see sections 3.1.4.5, 3.1.4.6, 3.1.4.7, and 3.1.4.8.

```
typedef struct _longAndMultiString_Return {  
    long ReturnCode;  
    [range(0,65536)] unsigned long cBytes;  
    [unique] [size_is(cBytes)] byte *msz;  
} ListReaderGroups_Return, ListReaders_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. The value returned from the Smart Card Redirection call.

**cBytes:** The number of bytes in the **msz** array field.

**msz:** The meaning of this field is specific to the context (IOCTL) in which it is used.

Value	Meaning
SCARD_IOCTL_LISTREADERSA 0x00090028	ASCII multistring of readers on the system.
SCARD_IOCTL_LISTREADERSW 0x0009002C	Unicode multistring of readers on the system.
SCARD_IOCTL_LISTREADERGROUPSA 0x00090020	ASCII multistring of reader groups on the system.
SCARD_IOCTL_LISTREADERGROUPSW 0x00090024	Unicode multistring of reader groups on the system.

### 2.2.3.5 LocateCards\_Return and GetStatusChange\_Return

The LocateCards\_Return and GetStatusChange\_Return structures are used to obtain the results on those calls that return updated reader state information. (for more information, see sections 3.1.4.21, 3.1.4.22, 3.1.4.23, 3.1.4.24, 3.1.4.25, and 3.1.4.26).

```

typedef struct _LocateCards_Return {
    long ReturnCode;
    [range(0, 10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderState_Return *rgReaderStates;
} LocateCards_Return,
GetStatusChange_Return;

```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**cReaders:** The number of elements in the **rgReaderStates** field.

**rgReaderStates:** The current states of the readers being watched.

### 2.2.3.6 Control\_Return

The Control\_Return structure is used to obtain information from a Control\_Call (for more information, see section 3.1.4.37).

```

typedef struct _Control_Return {
    long ReturnCode;
    [range(0, 66560)] unsigned long cbOutBufferSize;
    [unique] [size_is(cbOutBufferSize)] byte *pvOutBuffer;
} Control_Return;

```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**cbOutBufferSize:** The number of bytes in the **pvOutBuffer** field.

**pvOutBuffer:** Contains the return data specific to the value of the Control\_Call structure.

### 2.2.3.7 Reconnect\_Return

The Reconnect\_Return structure is used to obtain return information from a Reconnect call (for more information, see section 3.1.4.36).

```

typedef struct Reconnect_Return {
    long ReturnCode;
    unsigned long dwActiveProtocol;
} Reconnect_Return;

```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**dwActiveProtocol:** A flag that indicates the established active protocol. For more information on acceptable values, see section 2.2.5 .

### 2.2.3.8 Connect\_Return

The Connect\_Return structure is used to obtain return information from a Connect call (for more information, see sections 3.1.4.28 and 3.1.4.29).

```

typedef struct _Connect_Return {
    long ReturnCode;
    REDIR_SCARDHANDLE hCard;
}

```

```
    unsigned long dwActiveProtocol;
} Connect_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**hCard:** A handle, as specified in section 2.2.1.2.

**dwActiveProtocol:** A value that indicates the active smart card transmission protocol. Possible values are specified in section 2.2.5.

### 2.2.3.9 State\_Return

The State\_Return structure defines return information about the state of the smart card reader (for more information, see section 3.1.4.40).

```
typedef struct _State_Return {
    long ReturnCode;
    unsigned long dwState;
    unsigned long dwProtocol;
    [range(0,36)] unsigned long cbAtrLen;
    [unique] [size_is(cbAtrLen)] byte *rgAtr;
} State_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**dwState:** The current state of the smart card in the Reader. Possible values are specified in section 2.2.4.

**dwProtocol:** The current protocol, if any. Possible values are specified in section 2.2.5.

**cbAtrLen:** The number of bytes in the **rgAtr** field.

**rgAtr:** A pointer to a buffer that receives the ATR string from the currently inserted card, if available.

### 2.2.3.10 Status\_Return

The Status\_Return structure defines return information about the status of the smart card reader (for more information, see sections 3.1.4.33 and 3.1.4.34).

```
typedef struct _Status_Return {
    long ReturnCode;
    [range(0,65536)] unsigned long cBytes;
    [unique] [size_is(cBytes)] byte *mszReaderNames;
    unsigned long dwState;
    unsigned long dwProtocol;
    byte pbAtr[32];
    [range(0,32)] unsigned long cbAtrLen;
} Status_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**cBytes:** The number of bytes in the **mszReaderNames** field.

**mszReaderNames:** A multistring containing the names that the reader is known by. The value of this is dependent on the context (IOCTL) that it is used.

Value	Meaning
SCARD_IOCTL_STATUSA 0x000900C8	ASCII multistring
SCARD_IOCTL_STATUSW 0x000900CC	Unicode multistring

**dwState:** The current state of the smart card in the reader. Possible values are specified in section 2.2.4.

**dwProtocol:** The current protocol, if any. Possible values are specified in section 2.2.5.

**pbAtr:** A pointer to a buffer that receives the ATR string from the currently inserted card, if available.

**cbAtrLen:** The number of bytes in the ATR string.

### 2.2.3.11 Transmit\_Return

The Transmit\_Return structure defines return information from a smart card after a Transmit call (for more information, see section 3.1.4.35).

```
typedef struct _Transmit_Return {
    long ReturnCode;
    [unique] SCardIO_Request *pioRecvPci;
    [range(0, 66560)] unsigned long cbRecvLength;
    [unique] [size_is(cbRecvLength)] byte *pbRecvBuffer;
} Transmit_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**pioRecvPci:** The protocol header structure for the instruction, followed by a buffer in which to receive any returned protocol control information (PCI) that is specific to the protocol in use. If this field is NULL, a protocol header MUST NOT be returned.

**cbRecvLength:** The size, in bytes, of the **pbRecvBuffer** field.

**pbRecvBuffer:** The data returned from the card.

### 2.2.3.12 GetAttrib\_Return

The GetAttrib\_Return structure defines attribute information from a smart card reader (for more information, see section 3.1.4.38).

```
typedef struct _GetAttrib_Return {
    long ReturnCode;
    [range(0, 65536)] unsigned long cbAttrLen;
    [unique] [size_is(cbAttrLen)] byte *pbAttr;
} GetAttrib_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**cbAttrLen:** The number of bytes in the **pbAttr** field.

**pbAttr:** A pointer to an array that contains any values returned from the corresponding call.



### 2.2.3.13 GetTransmitCount\_Return

The GetTransmitCount\_Return structure defines the number of transmit calls that were performed on the smart card reader (for more information, see section 3.1.4.41).

```
typedef struct _GetTransmitCount_Return {
    long ReturnCode;
    unsigned long cTransmitCount;
} GetTransmitCount_Return;
```

**ReturnCode:** HRESULT or Win32 Error code. Zero indicates success; any other value indicates failure.

**cTransmitCount:** The field specifies the number of successful Transmit calls (for more information, see section 3.1.4.35) performed on the reader since it was introduced to the system.

### 2.2.3.14 GetReaderIcon\_Return

The GetReaderIcon\_Return structure is used to obtain the data that corresponds to the lookup item requested in the **GetReaderIcon\_Call** as specified in section 2.2.2.31. For more information, see section 3.1.4.48.

```
typedef struct _GetReaderIcon_Return {
    long ReturnCode;
    [range(0, 4194304)] unsigned long cbDataLen;
    [unique, size_is(cbDataLen)] byte* pbData;
} GetReaderIcon_Return;
```

**ReturnCode:** HRESULT or Win32 error code. Zero indicates success; any other value indicates failure.

**cbDataLen:** The number of bytes in the **pbData** field.

**pbData:** The value of the lookup item.

### 2.2.3.15 GetDeviceTypeId\_Return

The GetDeviceTypeId\_Return structure is used to obtain the data that corresponds to the lookup item requested in **GetDeviceTypeId\_Call** as specified in section 2.2.2.32. For more information, see section 3.1.4.47.

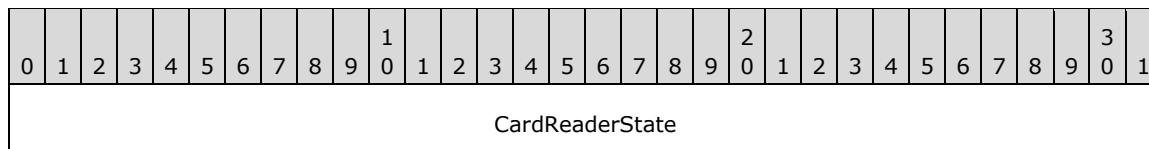
```
typedef struct _GetDeviceTypeId_Return {
    long ReturnCode;
    unsigned long dwDeviceId;
} GetDeviceTypeId_Return;
```

**ReturnCode:** HRESULT or Win32 error code. Zero indicates success; any other value indicates failure.

**dwDeviceId:** The value of the lookup item.

## 2.2.4 Card/Reader State

The following represents the current state of the smart card reader according to Smart Cards for Windows.

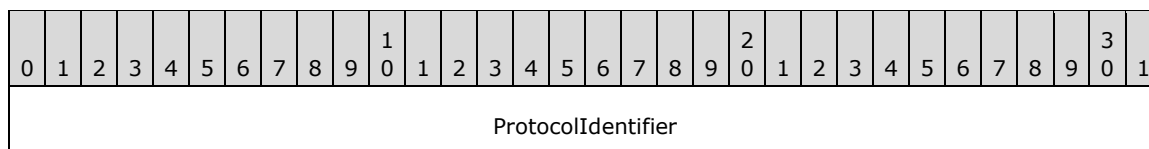


**CardReaderState (4 bytes):** One of the following values.

Value	Meaning
SCARD_UNKNOWN 0x00000000	The current state of the reader is unknown.
SCARD_ABSENT 0x00000001	There is no card in the reader.
SCARD_PRESENT 0x00000002	There is a card in the reader but it has not been moved into position for use.
SCARD_SWALLOWED 0x00000003	There is a card in the reader in position for use. The card is not powered.
SCARD_POWERED 0x00000004	There is power being applied to the card but the mode of the card is unknown.
SCARD_NEGOTIABLE 0x00000005	The card has been reset and is awaiting PTS negotiation.
SCARD_SPECIFICMODE 0x00000006	The card has been reset and specific communication protocols have been established.

## 2.2.5 Protocol Identifier

A Protocol Identifier.



**ProtocolIdentifier (4 bytes):** This field MUST have a value from Table A which is logically OR'ed with a value from Table B.

**Table A**

Value	Meaning
SCARD_PROTOCOL_UNDEFINED 0x00000000	No transmission protocol is active.
SCARD_PROTOCOL_T0 0x00000001	Transmission protocol 0 (T=0) is active. It is the asynchronous half-duplex character transmission protocol.
SCARD_PROTOCOL_T1 0x00000002	Transmission protocol 1 (T=1) is active. It is the asynchronous half-duplex block transmission protocol.
SCARD_PROTOCOL_Tx	Bitwise OR combination of both of the two International Standards Organization (ISO) transmission protocols SCARD_PROTOCOL_T0

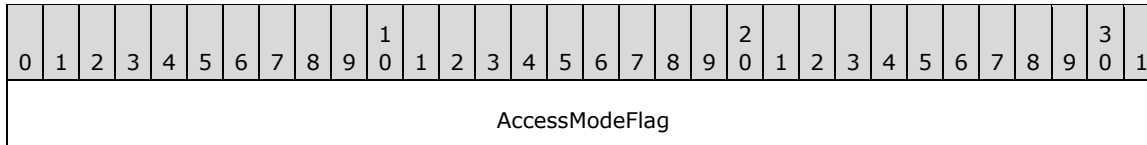
Value	Meaning
0x00000003	and SCARD_PROTOCOL_T1. This value can be used as a bitmask.
SCARD_PROTOCOL_RAW 0x00010000	Transmission protocol raw is active. The data from the smart card is raw and does not conform to any transmission protocol.

**Table B**

Value	Meaning
SCARD_PROTOCOL_DEFAULT 0x80000000	A bitwise OR with this value forces the use of the default transmission parameters and card clock frequency.
SCARD_PROTOCOL_OPTIMAL 0x00000000	Optimal transmission parameters and card clock frequency MUST be used. This flag is considered the default. No actual value is defined for this flag; it is there for compatibility with [PCSC5] section 3.1.3.

## 2.2.6 Access Mode Flags

Access mode flags provide possible values for applications to connect to the smart card.

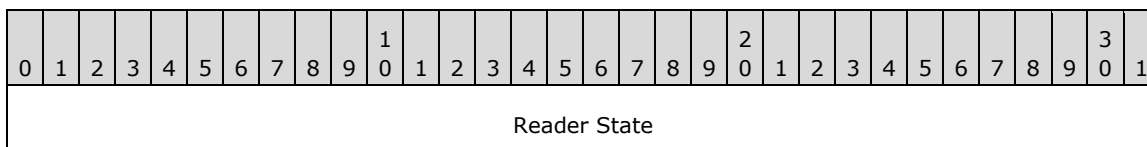


**AccessModeFlag (4 bytes):** One of the following possible values:

Value	Meaning
SCARD_SHARE_EXCLUSIVE 0x00000001	This application is not willing to share this smart card with other applications.
SCARD_SHARE_SHARED 0x00000002	This application is willing to share this smart card with other applications.
SCARD_SHARE_DIRECT 0x00000003	This application demands direct control of the smart card reader; therefore, it is not available to other applications.

## 2.2.7 Reader State

The Reader State packet has a sub-structure as shown in the following table.



**Reader State (4 bytes):** Both the **dwCurrentState** field and the **dwEventState** field, found in the ReaderState\_Common\_Call (section 2.2.1.5) and ReaderState\_Return (section 2.2.1.11) structures, consist of the following two subfields.

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
Count																State															

**Count (2 bytes):** The contents of this field depend on the value of the associated reader name. If the reader name (for more information, see sections 2.2.1.6 and 2.2.1.7 for the **szReader** field) is \\?PnP?\Notification, then **Count** is a count of the number of readers installed on the system and all bits except SCARD\_STATE\_CHANGED in **State** MUST be zero. Otherwise, **Count** is a count of the number of times a card has been inserted and/or removed from the smart card reader being monitored.

**State (2 bytes):** The state of a reader. The value MUST be according to the following table.

Value	Meaning
SCARD_STATE_UNAWARE 0x0000	The application requires the current state but does not know it. The use of this value results in an immediate return from state transition monitoring services.
SCARD_STATE_IGNORE 0x0001	The application requested that this reader be ignored. If this bit is set in the <b>dwCurrentState</b> field of a ReaderState_Common_Call structure, other bits MUST NOT be set in the <b>dwEventState</b> field of the corresponding ReaderState_Return structure.
SCARD_STATE_CHANGED 0x0002	There is a difference between the state believed by the application, and the state known by Smart Cards for Windows.
SCARD_STATE_UNKNOWN 0x0004	The reader name is not recognized by Smart Cards for Windows. If this bit is set in the <b>dwEventState</b> field of the ReaderState_Return structure, both SCARD_STATE_IGNORE and SCARD_STATE_CHANGED values MUST be set. This bit SHOULD NOT be set in the <b>dwCurrentState</b> field of a ReaderState_Common_Call structure.
SCARD_STATE_UNAVAILABLE 0x0008	The actual state of this reader is not available. If this bit is set, all of the following bits MUST be clear.
SCARD_STATE_EMPTY 0x0010	There is no card in the reader. If this bit is set, all of the following bits MUST be clear.
SCARD_STATE_PRESENT 0x0020	There is a card in the reader.
SCARD_STATE_ATRMATCH 0x0040	There is a card in the reader with an ATR that matches one of the target cards. If this bit is set, SCARD_STATE_PRESENT MUST be set.
SCARD_STATE_EXCLUSIVE 0x0080	The card in the reader is allocated for exclusive use by another application. If this bit is set, SCARD_STATE_PRESENT MUST be set.
SCARD_STATE_INUSE 0x0100	The card in the reader is in use by one or more other applications, but it can be connected to in shared mode. If this bit is set, SCARD_STATE_PRESENT MUST be set.
SCARD_STATE_MUTE 0x0200	The card in the reader is unresponsive or is not supported by the reader or software.
SCARD_STATE_UNPOWERED 0x0400	This implies that the card in the reader has not been turned on.

## 2.2.8 Return Code

The following Smart Card Facility Codes for Windows-specific return codes MAY be returned by the protocol server to the protocol client and are of the data type NTSTATUS, with the **sev** field set to STATUS\_SEVERITY\_WARNING (0x2) and the reserved bit (**N**) set to 0.

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
ReturnCode																															

**ReturnCode (4 bytes):** One of the following return codes:

Value	Meaning
SCARD_S_SUCCESS 0x00000000	No error has occurred.
SCARD_F_INTERNAL_ERROR 0x80100001	An internal consistency check failed.
SCARD_E_CANCELLED 0x80100002	The action was canceled by a Cancel request.
SCARD_E_INVALID_HANDLE 0x80100003	The supplied handle was invalid.
SCARD_E_INVALID_PARAMETER 0x80100004	One or more of the supplied parameters could not be properly interpreted.
SCARD_E_INVALID_TARGET 0x80100005	Registry startup information is missing or invalid.
SCARD_E_NO_MEMORY 0x80100006	Not enough memory available to complete this command.
SCARD_F_WAITED_TOO_LONG 0x80100007	An internal consistency timer has expired.
SCARD_E_INSUFFICIENT_BUFFER 0x80100008	The data buffer to receive returned data is too small for the returned data.
SCARD_E_UNKNOWN_READER 0x80100009	The specified reader name is not recognized.
SCARD_E_TIMEOUT 0x8010000A	The user-specified time-out value has expired.
SCARD_E_SHARING_VIOLATION 0x8010000B	The smart card cannot be accessed because of other connections outstanding.
SCARD_E_NO_SMARTCARD 0x8010000C	The operation requires a smart card, but no smart card is currently in the device.
SCARD_E_UNKNOWN_CARD 0x8010000D	The specified smart card name is not recognized.
SCARD_E_CANT_DISPOSE	The system could not dispose of the media in the requested

<b>Value</b>	<b>Meaning</b>
0x8010000E	manner.
SCARD_E_PROTO_MISMATCH 0x8010000F	The requested protocols are incompatible with the protocol currently in use with the smart card.
SCARD_E_NOT_READY 0x80100010	The reader or smart card is not ready to accept commands.
SCARD_E_INVALID_VALUE 0x80100011	One or more of the supplied parameters values could not be properly interpreted.
SCARD_E_SYSTEM_CANCELLED 0x80100012	The action was canceled by the system, presumably to log off or shut down.
SCARD_F_COMM_ERROR 0x80100013	An internal communications error has been detected.
SCARD_F_UNKNOWN_ERROR 0x80100014	An internal error has been detected, but the source is unknown.
SCARD_E_INVALID_ATR 0x80100015	An ATR obtained from the registry is not a valid ATR string.
SCARD_E_NOT_TRANSACTED 0x80100016	An attempt was made to end a non-existent transaction.
SCARD_E_READER_UNAVAILABLE 0x80100017	The specified reader is not currently available for use.
SCARD_P_SHUTDOWN 0x80100018	The operation has been stopped to allow the server application to exit.
SCARD_E_PCI_TOO_SMALL 0x80100019	The PCI Receive buffer was too small.
SCARD_E_ICC_INSTALLATION 0x80100020	No primary provider can be found for the smart card.
SCARD_E_ICC_CREATEORDER 0x80100021	The requested order of object creation is not supported.
SCARD_E_UNSUPPORTED_FEATURE 0x80100022	This smart card does not support the requested feature.
SCARD_E_DIR_NOT_FOUND 0x80100023	The specified directory does not exist in the smart card.
SCARD_E_FILE_NOT_FOUND 0x80100024	The specified file does not exist in the smart card.
SCARD_E_NO_DIR 0x80100025	The supplied path does not represent a smart card directory.
SCARD_E_READER_UNSUPPORTED 0x8010001A	The reader device driver does not meet minimal requirements for support.
SCARD_E_DUPLICATE_READER	The reader device driver did not produce a unique reader

<b>Value</b>	<b>Meaning</b>
0x8010001B	name.
SCARD_E_CARD_UNSUPPORTED 0x8010001C	The smart card does not meet minimal requirements for support.
SCARD_E_NO_SERVICE 0x8010001D	Smart Cards for Windows is not running.
SCARD_E_SERVICE_STOPPED 0x8010001E	Smart Cards for Windows has shut down.
SCARD_E_UNEXPECTED 0x8010001F	An unexpected card error has occurred.
SCARD_E_NO_FILE 0x80100026	The supplied path does not represent a smart card file.
SCARD_E_NO_ACCESS 0x80100027	Access is denied to this file.
SCARD_E_WRITE_TOO_MANY 0x80100028	The smart card does not have enough memory to store the information.
SCARD_E_BAD_SEEK 0x80100029	There was an error trying to set the smart card file object pointer.
SCARD_E_INVALID_CHV 0x8010002A	The supplied PIN is incorrect.
SCARD_E_UNKNOWN_RES_MSG 0x8010002B	An unrecognized error code was returned from a layered component.
SCARD_E_NO_SUCH_CERTIFICATE 0x8010002C	The requested certificate does not exist.
SCARD_E_CERTIFICATE_UNAVAILABLE 0x8010002D	The requested certificate could not be obtained.
SCARD_E_NO_READERS_AVAILABLE 0x8010002E	Cannot find a smart card reader.
SCARD_E_COMM_DATA_LOST 0x8010002F	A communications error with the smart card has been detected. Retry the operation.
SCARD_E_NO_KEY_CONTAINER 0x80100030	The requested key container does not exist.
SCARD_E_SERVER_TOO_BUSY 0x80100031	Smart Cards for Windows is too busy to complete this operation.
SCARD_E_PIN_CACHE_EXPIRED 0x80100032	The smart card PIN cache has expired.
SCARD_E_NO_PIN_CACHE 0x80100033	The smart card PIN cannot be cached.
SCARD_E_READ_ONLY_CARD	The smart card is read-only and cannot be written to.

<b>Value</b>	<b>Meaning</b>
0x80100034	
SCARD_W_UNSUPPORTED_CARD 0x80100065	The reader cannot communicate with the smart card due to ATR configuration conflicts.
SCARD_W_UNRESPONSIVE_CARD 0x80100066	The smart card is not responding to a reset.
SCARD_W_UNPOWERED_CARD 0x80100067	Power has been removed from the smart card, so that further communication is impossible.
SCARD_W_RESET_CARD 0x80100068	The smart card has been reset, so any shared state information is invalid.
SCARD_W_REMOVED_CARD 0x80100069	The smart card has been removed, so that further communication is impossible.
SCARD_W_SECURITY_VIOLATION 0x8010006A	Access was denied because of a security violation.
SCARD_W_WRONG_CHV 0x8010006B	The card cannot be accessed because the wrong PIN was presented.
SCARD_W_CHV_BLOCKED 0x8010006C	The card cannot be accessed because the maximum number of PIN entry attempts has been reached.
SCARD_W_EOF 0x8010006D	The end of the smart card file has been reached.
SCARD_W_CANCELLED_BY_USER 0x8010006E	The action was canceled by the user.
SCARD_W_CARD_NOT_AUTHENTICATED 0x8010006F	No PIN was presented to the smart card.
SCARD_W_CACHE_ITEM_NOT_FOUND 0x80100070	The requested item could not be found in the cache.
SCARD_W_CACHE_ITEM_STALE 0x80100071	The requested cache item is too old and was deleted from the cache.
SCARD_W_CACHE_ITEM_TOO_BIG 0x80100072	The new cache item exceeds the maximum per-item size defined for the cache.



## 3 Protocol Details

The following sections specify details of the Remote Desktop Protocol: Smart Card Virtual Channel Extension, including abstract data models, interface method syntax, and message processing rules.

### 3.1 Protocol Server Details

#### 3.1.1 Abstract Data Model

This section describes a conceptual model of a 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 provided that their external behavior is consistent with that described in this document.

The protocol server relies on an implementation of Smart Cards for Windows.

The following state **MUST** be kept by this protocol:

**dwDeviceId**: The device id assigned by Remote Desktop Protocol: File System Virtual Channel Extension that identifies this protocol.

**rgSCardContextList**: List of contexts opened by the protocol server.

#### 3.1.2 Timers

None.

#### 3.1.3 Initialization

Initialization is triggered by the Remote Desktop Protocol: File System Virtual Channel Extension when it enumerates all pre-logon devices. At this time, TS client initialization is performed.

If the TS server operating system version is earlier than 5.1, the device is not announced to the TS server.

The **dwDeviceId** field **MUST** be set to the device Id selected by Remote Desktop Protocol: File System Virtual Channel Extension, and **rgSCardContextList** **MUST** be set to the empty list.

#### 3.1.4 Message Processing Events and Sequencing Rules

Only messages of type DR\_CONTROL\_REQ and DR\_CONTROL\_RSP (as specified in [MS-RDPEFS] sections 2.2.1.4.5 and 2.2.1.5.5, respectively) are valid for this protocol. All other messages **MUST** be processed according to the Remote Desktop Protocol: File System Virtual Channel Extension.

Only the control codes specified in the IOCTL Processing Rules in the following table are valid. Invalid packets **MUST** be dropped without a reply.

Function number	Value for IoControlCode	IRP_MJ_DEVICE_CONTROL request	Input packet, Output packet
5	0x00090014	SCARD_IOCTL_ESTABLISHCONTEXT	EstablishContext_Call (section 2.2.2.1), EstablishContext_Return (section 2.2.3.2)

Function number	Value for IoControlCode	IRP_MJ_DEVICE_CONTROL request	Input packet, Output packet
6	0x00090018	SCARD_IOCTL_RELEASECONTEXT	Context_Call (section 2.2.2.2), Long_Return (section 2.2.3.3)
7	0x0009001C	SCARD_IOCTL_ISVALIDCONTEXT	Context_Call (section 2.2.2.2), Long_Return (section 2.2.3.3)
8	0x00090020	SCARD_IOCTL_LISTREADERGROUPSA	ListReaderGroups_Call (section 2.2.2.3), ListReaderGroups_Return (section 2.2.3.4)
9	0x00090024	SCARD_IOCTL_LISTREADERGROUPSW	ListReaderGroups_Call (section 2.2.2.3), ListReaderGroups_Return (section 2.2.3.4)
10	0x00090028	SCARD_IOCTL_LISTREADERSA	ListReaders_Call (section 2.2.2.4), ListReaders_Return (section 2.2.3.4)
11	0x0009002C	SCARD_IOCTL_LISTREADERSW	ListReaders_Call (section 2.2.2.4), ListReaders_Return (section 2.2.3.4)
20	0x00090050	SCARD_IOCTL_INTRODUCEREADERGRO UPA	ContextAndStringA_Call (section 2.2.2.5), Long_Return (section 2.2.3.3)
21	0x00090054	SCARD_IOCTL_INTRODUCEREADERGRO UPW	ContextAndStringW_Call (section 2.2.2.6), Long_Return (section 2.2.3.3)
22	0x00090058	SCARD_IOCTL_FORGETREADERGROUPA	ContextAndStringA_Call (section 2.2.2.5), Long_Return (section 2.2.3.3)
23	0x0009005C	SCARD_IOCTL_FORGETREADERGROUP W	ContextAndStringW_Call (section 2.2.2.6), Long_Return (section 2.2.3.3)
24	0x00090060	SCARD_IOCTL_INTRODUCEREADERA	ContextAndTwoStringA_Call (section 2.2.2. .7), Long_Return (section 2.2.3.3)
25	0x00090064	SCARD_IOCTL_INTRODUCEREADERW	ContextAndTwoStringW_Call (section 2.2. 2.8), Long_Return (section 2.2.3.3)
26	0x00090068	SCARD_IOCTL_FORGETREADERA	ContextAndStringA_Call (section 2.2.2.5), Long_Return (section 2.2.3.3)
27	0x0009006C	SCARD_IOCTL_FORGETREADERW	ContextAndStringW_Call (section 2.2.2.6), Long_Return (section 2.2.3.3)
28	0x00090070	SCARD_IOCTL_ADDREADERTOGROUPA	ContextAndTwoStringA_Call (section 2.2.2. .7), Long_Return (section 2.2.3.3)
29	0x00090074	SCARD_IOCTL_ADDREADERTOGROUPW	ContextAndTwoStringW_Call (section 2.2. 2.8), Long_Return (section 2.2.3.3)
30	0x00090078	SCARD_IOCTL_REMOVEREADERFROMG ROUPA	ContextAndTwoStringA_Call (section 2.2.2. .7), Long_Return (section 2.2.3.3)
31	0x0009007C	SCARD_IOCTL_REMOVEREADERFROMG ROUPW	ContextAndTwoStringW_Call (section 2.2. 2.8), Long_Return (section 2.2.3.3)
38	0x00090098	SCARD_IOCTL_LOCATECARDSA	LocateCardsA_Call (section 2.2.2.9), LocateCards_Return (section 2.2.3.5)
39	0x0009009C	SCARD_IOCTL_LOCATECARDSW	LocateCardsW_Call (section 2.2.2.10), LocateCards_Return (section 2.2.3.5)

Function number	Value for IoControlCode	IRP_MJ_DEVICE_CONTROL request	Input packet, Output packet
40	0x000900A0	SCARD_IOCTL_GETSTATUSCHANGEA	GetStatusChangeA_Call (section 2.2.2.11) , GetStatusChange_Return (section 2.2.3.5)
41	0x000900A4	SCARD_IOCTL_GETSTATUSCHANGEW	GetStatusChangeW_Call (section 2.2.2.12) , GetStatusChange_Return (section 2.2.3.5)
42	0x000900A8	SCARD_IOCTL_CANCEL	Context_Call (section 2.2.2.2), Long_Return (section 2.2.3.3)
43	0x000900AC	SCARD_IOCTL_CONNECTA	ConnectA_Call (section 2.2.2.13), Connect_Return (section 2.2.3.8)
44	0x000900B0	SCARD_IOCTL_CONNECTW	ConnectW_Call (section 2.2.2.14), Connect_Return (section 2.2.3.8)
45	0x000900B4	SCARD_IOCTL_RECONNECT	Reconnect_Call (section 2.2.2.15), Reconnect_Return (section 2.2.3.7)
46	0x000900B8	SCARD_IOCTL_DISCONNECT	HCardAndDisposition_Call (section 2.2.2.16), Long_Return (section 2.2.3.3)
47	0x000900BC	SCARD_IOCTL_BEGINTRANSACTION	HCardAndDisposition_Call (section 2.2.2.16), Long_Return (section 2.2.3.3)
48	0x000900C0	SCARD_IOCTL_ENDTRANSACTION	HCardAndDisposition_Call (section 2.2.2.16), Long_Return (section 2.2.3.3)
49	0x000900C4	SCARD_IOCTL_STATE	State_Call (section 2.2.2.17), State_Return (section 2.2.3.9)
50	0x000900C8	SCARD_IOCTL_STATUSA	Status_Call (section 2.2.2.18), Status_Return (section 2.2.3.10)
51	0x000900CC	SCARD_IOCTL_STATUSW	Status_Call (section 2.2.2.18), Status_Return (section 2.2.3.10)
52	0x000900D0	SCARD_IOCTL_TRANSMIT	Transmit_Call (section 2.2.2.19), Transmit_Return (section 2.2.3.11)
53	0x000900D4	SCARD_IOCTL_CONTROL	Control_Call (section 2.2.2.20), Control_Return (section 2.2.3.6)
54	0x000900D8	SCARD_IOCTL_GETATTRIB	GetAttrib_Call (section 2.2.2.21), GetAttrib_Return (section 2.2.3.12)
55	0x000900DC	SCARD_IOCTL_SETATTRIB	SetAttrib_Call (section 2.2.2.22), Long_Return (section 2.2.3.3)
56	0x000900E0	SCARD_IOCTL_ACCESSSTARTEDEVENT	ScardAccessStartedEvent_Call (section 2.2.2.30), Long_Return (section 2.2.3.3)
57	0x000900E4	SCARD_IOCTL_RELEASESTARTEDEVENT	Not used.
58	0x000900E8	SCARD_IOCTL_LOCATECARDSBYATRA	LocateCardsByATRA_Call (section 2.2.2.23), LocateCards_Return (section 2.2.3.5)
59	0x000900EC	SCARD_IOCTL_LOCATECARDSBYATRW	LocateCardsByATRW_Call (section 2.2.2.2)

Function number	Value for IoControlCode	IRP_MJ_DEVICE_CONTROL request	Input packet, Output packet
			4), LocateCards_Return (section 2.2.3.5)
60	0x000900F0	SCARD_IOCTL_READCACHEA	ReadCacheA_Call (section 2.2.2.25), ReadCache_Return (section 2.2.3.1)
61	0x000900F4	SCARD_IOCTL_READCACHEW	ReadCacheW_Call (section 2.2.2.26), ReadCache_Return (section 2.2.3.1)
62	0x000900F8	SCARD_IOCTL_WRITECACHEA	WriteCacheA_Call (section 2.2.2.27), Long_Return (section 2.2.3.3)
63	0x000900FC	SCARD_IOCTL_WRITECACHEW	WriteCacheW_Call (section 2.2.2.28), Long_Return (section 2.2.3.3)
64	0x00090100	SCARD_IOCTL_GETTRANSMITCOUNT	GetTransmitCount_Call (section 2.2.2.29), GetTransmitCount_Return (section 2.2.3.13)
65	0x00090104	SCARD_IOCTL_GETREADERICON	GetReaderIcon_Call (section 2.2.2.31), GetReaderIcon_Return (section 2.2.3.14)
66	0x00090108	SCARD_IOCTL_GETDEVICETYPEID	GetDeviceTypeId_Call (section 2.2.2.32), GetDeviceTypeId_Return (section 2.2.3.15)

The TS client MUST be able to process multiple requests simultaneously within the limits of its resources.

Any errors from the Smart Cards for Windows layer MUST be transferred to the TS server and MUST NOT be modified by the TS client. No exceptions are thrown in this protocol.

The following steps MUST be performed on each call packet received:

1. The IoControlCode MUST be present, as specified in the preceding IOCTL Processing Rules table, for the specific protocol version implemented. <2>
2. The input data type is interpreted according to the IOCTL Processing Rules table. The data MUST be decoded as specified in [MS-RPCE] section 2.2.6.
3. Processing MUST be performed according to the corresponding section that follows. On success, it MUST return a structure as specified in the preceding IOCTL Processing Rules table.
4. If the protocol encounters problems decoding the input or encoding the results, then DR\_DEVICE\_IOCTLCOMPLETION.IoStatus (as specified in [MS-RDPEFS] section 2.2.1.5) MUST be set to an NTSTATUS code (as specified in [MS-ERREF] section 2.3), the most common of which appear in the following table.

Return value/code	Description
STATUS_NO_MEMORY 0xC0000017	Not enough virtual memory or paging file quota is available to complete the specified operation.
STATUS_UNSUCCESSFUL 0xC0000001	The requested operation was unsuccessful.

Return value/code	Description
STATUS_BUFFER_TOO_SMALL 0xC0000023	The buffer is too small to contain the entry. No information has been written to the buffer.

5. On error, DR\_DEVICE\_IOCTLCOMPLETION.Parameters.DeviceIOControl.OutputBufferLength MUST be set to zero and DR\_DEVICE\_IOCTLCOMPLETION.Parameters.DeviceIOControl.OutputBuffer MUST set to NULL.
6. Otherwise, DR\_DEVICE\_IOCTLCOMPLETION.IOStatus MUST be set to 0 (STATUS\_SUCCESS) and DR\_DEVICE\_IOCTLCOMPLETION.Parameters.DeviceIOControl.OutputBuffer MUST contain an encoding of the structure (as specified in the preceding Message Processing Events and Sequencing Rules IOCTL Table) as specified in [MS-RPCE] section 2.2.6. DR\_DEVICE\_IOCTLCOMPLETION.Parameters.DeviceIOControl.OutputBufferLength is the length of the data.
7. The return packet is then sent according to Remote Desktop Protocol: File System Virtual Channel Extension.

#### **3.1.4.1 SCARD\_IOCTL\_ESTABLISHCONTEXT (IOCTL 0x00090014)**

Establish Context creates a new Smart Cards for Windows context specified for use in subsequent communication with Smart Cards for Windows.

Return Values: This method sets EstablishContext\_Return.ReturnCode to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

If the call is successful, EstablishContext\_Return.Context MUST be added to the rgSCardContextList list maintained by this client.

#### **3.1.4.2 SCARD\_IOCTL\_RELEASECONTEXT (IOCTL 0x00090018)**

Release Context releases a previously established Smart Cards for Windows context as specified in section 3.1.4.1. The context MUST exist in **rgSCardContextList**.

Return Values: This method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

If the call is successful, Context\_Call.Context (for more information, see section 2.2.2.2) is removed from rgSCardContextList.

#### **3.1.4.3 SCARD\_IOCTL\_ISVALIDCONTEXT (IOCTL 0x0009001C)**

Is Valid Context checks if a previously established Smart Cards for Windows context from SCARD\_IOCTL\_ESTABLISHCONTEXT is still valid. For this call to succeed, Context\_Call.Context (for more information, see section 2.2.2.2) MUST exist in rgSCardContextList and the Smart Cards for Windows communication channel MUST still be present.

Return Values: This method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.4 SCARD\_IOCTL\_ACCESSSTARTEDEVENT (IOCTL 0x000900E0)**

Access Started Event waits until Smart Cards for Windows is running.

Return Values: This method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS if Smart Cards for Windows is running; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.5 SCARD\_IOCTL\_LISTREADERGROUPSA (IOCTL 0x00090020)**

The ASCII version List Reader Groups returns the reader groups known to Smart Cards for Windows. ListReaderGroups\_Return is constructed according to ListReaderGroups\_Return and ListReaders\_Return and the information in ListReaderGroups\_Call.

Return Values: This method sets ListReaderGroups\_Return.ReturnCode (for more information, see section 2.2.3.4) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.6 SCARD\_IOCTL\_LISTREADERGROUPSW (IOCTL 0x00090024)**

The Unicode version List Reader Groups returns the reader groups known to Smart Cards for Windows. ListReaderGroups\_Return is constructed according to ListReaderGroups\_Return and ListReaders\_Return and the information in ListReaderGroups\_Call.

Return Values: This method sets ListReaderGroups\_Return.ReturnCode (for more information, see section 2.2.3.4) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.7 SCARD\_IOCTL\_LISTREADERSA (IOCTL 0x00090028)**

The ASCII version of List Readers returns the smart card readers known to Smart Cards for Windows. ListReaders\_Return is constructed according to ListReaderGroups\_Return and ListReaders\_Return and ListReaders\_Call.

Return Values: The method sets ListReaders\_Return.ReturnCode (for more information, see section 2.2.3.4) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.8 SCARD\_IOCTL\_LISTREADERSW (IOCTL 0x0009002C)**

The Unicode version of List Readers returns the smart card readers known to Smart Cards for Windows. ListReaders\_Return is constructed according to ListReaderGroups\_Return and ListReaders\_Return and ListReaders\_Call.

Return Values: The method sets ListReaders\_Return.ReturnCode (for more information, see section 2.2.3.4) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.9 SCARD\_IOCTL\_INTRODUCEREADERGROUPA (IOCTL 0x00090050)**

The ASCII version of Introduce Reader Group adds the reader group specified in ContextAndStringA\_Call.sz (for more information, see section 2.2.2.5) to the list of reader groups known to Smart Cards for Windows.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.10 SCARD\_IOCTL\_INTRODUCEREADERGROUPW (IOCTL 0x00090054)**

The Unicode version of Introduce Reader Group adds the reader group specified in ContextAndStringW\_Call.sz (for more information, see section 2.2.2.6) to the list of reader groups known to Smart Cards for Windows.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.11 SCARD\_IOCTL\_FORGETREADERGROUPA (IOCTL 0x00090058)**

The ASCII version of Forget Reader Group removes the reader group specified in ContextAndStringA\_Call.sz (for more information, see section 2.2.2.5) from the list of reader groups known to the Smart Cards for Windows.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.12 SCARD\_IOCTL\_FORGETREADERGROUPW (IOCTL 0x0009005C)**

The Unicode version of Forget Reader Group removes the reader group specified in ContextAndStringW\_Call.sz (for more information, see section 2.2.2.6) from the list of reader groups known to Smart Cards for Windows.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.13 SCARD\_IOCTL\_INTRODUCEREADERA (IOCTL 0x00090060)**

The ASCII version of Introduce Reader adds the device name specified in ContextAndTwoStringA\_Call.sz2 (for more information, see section 2.2.2.7) to the smart card reader specified in ContextAndTwoStringA\_Call.sz1.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.14 SCARD\_IOCTL\_INTRODUCEREADERW (IOCTL 0x00090064)**

The Unicode version of Introduce Reader adds the device name specified in ContextAndTwoStringW\_Call.sz2 (for more information, see section 2.2.2.8) to the smart card reader specified in ContextAndTwoStringW\_Call.sz1.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.15 SCARD\_IOCTL\_FORGETREADERA (IOCTL 0x00090068)**

The ASCII version of Forget Reader removes the smart card reader specified in ContextAndStringA\_Call.sz (for more information, see section 2.2.2.5) from the list of smart card readers known to Smart Cards for Windows.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.16 SCARD\_IOCTL\_FORGETREADERW (IOCTL 0x0009006C)**

The Unicode version of Forget Reader removes the smart card reader specified in ContextAndStringW\_Call.sz (for more information, see section 2.2.2.6) from the list of smart card readers known to Smart Cards for Windows.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.17 SCARD\_IOCTL\_ADDREADERTOGROUPA (IOCTL 0x00090070)**

The ASCII version of Add Reader to Group adds the smart card reader specified in ContextAndTwoStringA\_Call.sz2 (for more information, see section 2.2.2.7).

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.18 SCARD\_IOCTL\_ADDREADERTOGROUPW (IOCTL 0x00090074)**

The Unicode version of Add Reader to Group adds the smart card reader specified in ContextAndTwoStringW\_Call.sz2 (for more information, see section 2.2.2.8).

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.19 SCARD\_IOCTL\_REMOVEREADERFROMGROUPA (IOCTL 0x00090078)**

The ASCII version of Remove Reader From Group removes the smart card reader specified in ContextAndTwoStringA\_Call.sz2 (for more information, see section 2.2.2.7).

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.20 SCARD\_IOCTL\_REMOVEREADERFROMGROUPW (IOCTL 0x0009007C)**

The Unicode version of Remove Reader From Group removes the smart card reader specified in ContextAndTwoStringW\_Call.sz2 (for more information, see section 2.2.2.8).

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.21 SCARD\_IOCTL\_LOCATECARDSA (IOCTL 0x00090098)**

The ASCII version of Locate Cards searches the readers specified in LocateCardsA\_Call.msCards (for more information, see section 2.2.2.9). Unknown Card Types MUST be ignored. LocateCards\_Return is constructed according to LocateCards\_Return and GetStatusChange\_Return by using the information in LocateCardsA\_Call.



Return Values: The method sets LocateCards\_Return.ReturnCode (for more information, see section 2.2.3.5) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.22 SCARD\_IOCTL\_LOCATECARDSW (IOCTL 0x0009009C)**

The Unicode version of Locate Cards searches the readers specified in LocateCardsW\_Call.msxCards (for more information, see section 2.2.2.10). Unknown Card Types MUST be ignored. LocateCards\_Return is constructed according to LocateCards\_Return and GetStatusChange\_Return by using the information in LocateCardsW\_Call.

Return Values: The method sets LocateCards\_Return.ReturnCode to SCARD\_S\_SUCCESS on success; otherwise it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.23 SCARD\_IOCTL\_GETSTATUSCHANGEA (IOCTL 0x000900A0)**

The ASCII version of Get Status Change monitors the smart card readers specified in GetStatusChangeA\_Call.rgReaderStates (for more information, see section 2.2.2.11) MUST correctly represent the state of the Readers as known by Smart Cards for Windows.

Return Values: The method sets GetStatusChange\_Return.ReturnCode to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.24 SCARD\_IOCTL\_GETSTATUSCHANGEW (IOCTL 0x000900A4)**

The Unicode version of Get Status Change monitors the smart card readers specified in GetStatusChangeW\_Call.rgReaderStates (for more information, see section 2.2.2.12) MUST correctly represent the state of the readers as known by Smart Cards for Windows.

Return Values: The method sets GetStatusChange\_Return.ReturnCode to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.25 SCARD\_IOCTL\_LOCATECARDSBYATRA (IOCTL 0x000900E8)**

The ASCII version of Locate Cards By ATR searches the Readers specified in LocateCardsByATRA\_Call.rgAtrMasks (for more information, see section 2.2.2.23). Unknown card types MUST be ignored. LocateCards\_Return is constructed according to LocateCards\_Return and GetStatusChange\_Return by using the information in LocateCardsByATRA\_Call.

Return Values: The method sets LocateCards\_Return.ReturnCode (for more information, see section 2.2.3.5) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.26 SCARD\_IOCTL\_LOCATECARDSBYATRW (IOCTL 0x000900EC)**

The Unicode version of Locate Cards By ATR searches the readers specified in LocateCardsByATRW\_Call.rgAtrMasks (LocateCardsByATRW\_Call). Unknown Card Types MUST be ignored. LocateCards\_Return is constructed according to LocateCards\_Return and GetStatusChange\_Return by using the information in LocateCardsByATRW\_Call.

Return Values: The method sets LocateCards\_Return.ReturnCode (for more information, see section 2.2.3.5) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.27 SCARD\_IOCTL\_CANCEL (IOCTL 0x000900A8)**

The Cancel method MUST instruct Smart Cards for Windows to cancel any outstanding calls by using the context specified by Context\_Call.Context (for more information, see section 2.2.2.2).

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.28 SCARD\_IOCTL\_CONNECTA (IOCTL 0x000900AC)**

The ASCII version of Connect establishes a handle to a smart card reader. On success, Connect\_Return is initialized according to Control\_Return.

Return Values: The method sets the Connect\_Return.ReturnCode (for more information, see section 2.2.3.8) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.29 SCARD\_IOCTL\_CONNECTW (IOCTL 0x000900B0)**

The Unicode version of Connect establishes a smart card reader handle. On success, Connect\_Return is initialized according to Control\_Return and the caller is given a handle to execute additional methods on the reader.

Return Values: The method sets the Connect\_Return.ReturnCode (for more information, see section 2.2.3.8) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.30 SCARD\_IOCTL\_DISCONNECT (IOCTL 0x000900B8)**

The disconnect method releases a smart card reader handle that was acquired in ConnectA\_Call or ConnectW\_Call, using HCardAndDisposition\_Call.dwDisposition. After a successful call, The smart card reader handle is released and MUST be made available to the system.

Return Values: The method sets Long\_Return.ReturnCode to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.31 SCARD\_IOCTL\_BEGINTRANSACTION (IOCTL 0x000900BC)**

The Begin Transaction method locks a smart card reader for exclusive access for the specified smart card reader handle. If the caller is unable to receive exclusive access, this call MUST block until the request can be met.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.32 SCARD\_IOCTL\_ENDTRANSACTION (IOCTL 0x000900C0)**

The End Transaction method releases a smart card reader after being locked by a previously successful call to Begin Transaction (for more information, see section 3.1.4.31).

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.33 SCARD\_IOCTL\_STATUSA (IOCTL 0x000900C8)**

The ASCII version of the Status call returns the current state of the smart card reader and any smart card inserted. On success, Status\_Return MUST be initialized according to Status\_Return.

Return Values: The method sets Status\_Return.ReturnCode (for more information, see section 2.2.3.10) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.34 SCARD\_IOCTL\_STATUSW (IOCTL 0x000900CC)**

The Unicode version of the Status call returns the current state of the smart card reader and any smart card inserted. On success, Status\_Return MUST be initialized according to Status\_Return.

Return Values: The method sets Status\_Return.ReturnCode (for more information, see section 2.2.3.10) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.35 SCARD\_IOCTL\_TRANSMIT (IOCTL 0x000900D0)**

The Transmit function sends a command to a smart card inserted to the smart card reader associated with the smart card reader handle. On success, the command has been successfully sent to the card and the response has been placed in Transmit\_Return.

Return Values: The method sets Transmit\_Return.ReturnCode (for more information, see section 2.2.3.11) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.36 SCARD\_IOCTL\_RECONNECT (IOCTL 0x000900B4)**

The reconnect method re-establishes a smart card reader handle. On success, the handle is valid once again.

Return Values: The method sets Reconnect\_Return.ReturnCode (for more information, see section 2.2.3.7) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.37 SCARD\_IOCTL\_CONTROL (IOCTL 0x000900D4)**

The Control function sends a command to a smart card reader associated with the smart card reader handle. On success, the command has been successfully sent to the smart card reader and the response has been placed in Control\_Return.

Return Values: The method sets Control\_Return.ReturnCode (for more information, see section 2.2.3.6) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

### **3.1.4.38 SCARD\_IOCTL\_GETATTRIB (IOCTL 0x000900D8)**

The Get Attribute function requests an attribute of the smart card reader associated with the smart card reader handle. On success, the attribute is copied to GetAttrib\_Return.

Return Values: The method sets GetAttrib\_Return.ReturnCode (for more information, see section 2.2.3.12) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors

or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.39 SCARD\_IOCTL\_SETATTRIB (IOCTL 0x000900DC)**

The Set Attribute function changes the value of an attribute of the smart card reader associated with the smart card reader handle.

Return Values: The method sets Long\_Return.ReturnCode (for more information, see section 2.2.3.3) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.40 SCARD\_IOCTL\_STATE (IOCTL 0x000900C4)**

The State method returns the current state of the smart card reader and any smart card inserted. On success, Status\_Return MUST be initialized as specified in section 2.2.3.10.

Return Values: The method sets State\_Return.ReturnCode (for more information, see section 2.2.3.9) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.41 SCARD\_IOCTL\_GETTRANSMITCOUNT (IOCTL 0x00090100)**

The Get Transmit Count retrieves the number of times a successful Transmit method (for more information, see section 3.1.4.35) has been performed on the smart card reader. On success, GetTrasmitCount\_Return MUST be initialized as specified in section 2.2.3.13.

Return Values: The method sets State\_Return.ReturnCode (for more information, see section 2.2.3.9) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.42 SCARD\_IOCTL\_READCACHEA (IOCTL 0x000900F0)**

The ASCII version of Read Cache retrieves cached data for a specific smart card. Data is cached according to the smart card UUID (ReadCacheA\_Call.Common.CardIdentifier; for more information, see section 2.2.1.9), the Card Lookup Name (ReadCacheA\_Call.szLookupName; for more information, see section 2.2.2.25), and the freshness of the data (ReadCacheA\_Call.Common.FreshnessCounter; for more information, see section 2.2.1.9). All three MUST match in order for this call to be successful. On success, ReadCache\_Return MUST be initialized as specified in section 2.2.3.1.

Return Values: The method sets ReadCache\_Return.ReturnCode (for more information, see section 2.2.3.1) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.43 SCARD\_IOCTL\_READCACHEW (IOCTL 0x000900F4)**

The Unicode version of Read Cache retrieves cached data for a specific smart card in a Smart Cards for Windows cache. Data is cached according to the smart card UUID (ReadCacheA\_Call.Common.CardIdentifier; for more information, see section 2.2.1.9), the Card Lookup Name (ReadCacheW\_Call.szLookupName; for more information, see section 2.2.2.26), and the freshness of the data (ReadCacheW\_Call.Common.FreshnessCounter; for more information, see section 2.2.1.9). All three MUST match in order for this call to be successful. On success, ReadCache\_Return MUST be initialized as specified in section 2.2.3.1.

Return Values: The method sets ReadCache\_Return.ReturnCode (for more information, see section 2.2.3.1) to SCARD\_S\_SUCCESS on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from Winerror.h. No specialized error codes are associated with this method.

#### **3.1.4.44 SCARD\_IOCTL\_WRITECACHEA (IOCTL 0x000900F8)**

The ASCII version of Write Cache stores data for a specific smart card in a Smart Cards for Windows cache. Data is cached according to the smart card UUID (`ReadCacheA_Call.szLookupName`; for more information, see section 2.2.2.25), and the freshness of the data (`ReadCacheA_Call.Common.FreshnessCounter`).

**Return Values:** The method sets `Long_Return.ReturnCode` (for more information, see section 2.2.3.3) to `SCARD_S_SUCCESS` on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from `Winerror.h`. No specialized error codes are associated with this method

#### **3.1.4.45 SCARD\_IOCTL\_WRITECACHEW (IOCTL 0x000900FC)**

The Unicode version of Write Cache stores data for a specific smart card in a Smart Cards for Windows cache. Data is cached according to the smart card UUID (`ReadCacheA_Call.szLookupName`; for more information, see section 2.2.2.25), and the freshness of the data (`ReadCacheA_Call.Common.FreshnessCounter`).

**Return Values:** The method sets `Long_Return.ReturnCode` (for more information, see section 2.2.3.3) to `SCARD_S_SUCCESS` on success; otherwise, it sets one of the smart card-specific errors or one of the return codes from `Winerror.h`. No specialized error codes are associated with this method

#### **3.1.4.46 SCARD\_IOCTL\_RELEASETARTEDEVENT**

The `SCARD_IOCTL_RELEASETARTEDEVENT` IOCTL value is not used.

#### **3.1.4.47 SCARD\_IOCTL\_GETREADERICON (IOCTL 0x00090104)**

Get Reader Icon retrieves the icon from the INF file for a specific smart card reader name (for more information, see **GetReaderIcon\_Call.szReaderName**, section 2.2.2.31). On success, **GetReaderIcon\_Return.pbData** contains the icon; for more information, see section 2.2.3.14.

**Return Values:** This method sets **GetReaderIcon\_Return.ReturnCode** (for more information, see section 2.2.3.14) to `SCARD_S_SUCCESS` on success; otherwise, it sets one of the smart card-specific errors or another error code. No specialized error codes are associated with this method.

#### **3.1.4.48 SCARD\_IOCTL\_GETDEVICETYPEID (IOCTL 0x00090108)**

Get Device Type ID retrieves the device type from the INF file for a specific smart card reader name (**GetDeviceTypeId\_Call.szReaderName**; for more information, see section 2.2.2.32). On success, **GetDeviceTypeId\_Return.dwDeviceId** contains the device type ID; for more information, see section 2.2.3.15.

**Return Values:** This method sets **GetDeviceTypeId\_Return.ReturnCode** (for more information, see section 2.2.3.15) to `SCARD_S_SUCCESS` on success; otherwise, it sets one of the smart card-specific errors or another error code. No specialized error codes are associated with this method.

### **3.1.5 Timer Events**

None.

### **3.1.6 Other Local Events**

On protocol termination, the following actions are performed.

For each context in `rgSCardContextList`, `Cancel` is called causing all outstanding messages to be processed. After there are no more outstanding messages, `Release Context` is called on each context and the context **MUST** be removed from `rgSCardContextList`.

## 3.2 Protocol Client Details

### 3.2.1 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 provided that their external behavior is consistent with that described in this document.

The following state **MUST** be kept by this protocol:

**dwDeviceId**: device ID of smart card redirection device.

**rgOutstandingMessages**: Outstanding call packets have not received a return packet.

### 3.2.2 Timers

No timers are required.

### 3.2.3 Initialization

Initialization occurs when the protocol server sends a device-announce message according to Remote Desktop Protocol: File System Virtual Channel Extension. At that time, **dwDeviceId** **MUST** receive the unique device ID announced. The **rgOutstandingMessage** field **MUST** be set to the empty list.

### 3.2.4 Higher-Layer Triggered Events

None.

### 3.2.5 Message Processing Events and Sequencing Rules

#### 3.2.5.1 Sending Outgoing Messages

Messages are constructed according to Remote Desktop Protocol: File System Virtual Channel Extension as a device I/O control message on the redirected device **dwDeviceId**. The call packet **MUST** follow the format specified in IOCTL Processing Rules. The structure **MUST** be encoded as specified in [MS-RPCE] section 2. The output buffer length **SHOULD** be set to 2,048 bytes.

The message is sent to the protocol server by using a transport as specified in [MS-RDPEFS] section 2.1.

#### 3.2.5.2 Processing Incoming Replies

The following steps **MUST** be applied to each message when they are received.

If `IOStatus` is `STATUS_BUFFER_TOO_SMALL`, then the message **SHOULD** be retransmitted according to Sending Outgoing Messages, doubling the previously requested buffer length.

If `IOStatus` is zero, the corresponding `IoControlCode`-specific reply processing **MUST** be performed.

Otherwise, the call is considered a failure and the error MUST be propagated to the higher layer.

### **3.2.5.3 Messages**

#### **3.2.5.3.1 Sending EstablishContext Message**

IoControlCode MUST be set to SCARD\_IOCTL\_ESTABLISHCONTEXT.

EstablishContext\_Call MUST be initialized as specified in section 2.2.2.1.

#### **3.2.5.3.2 Processing EstablishContext Reply**

The OutputBuffer MUST be decoded as EstablishContext\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.3 Sending ReleaseContext Message**

IoControlCode MUST be set to SCARD\_IOCTL\_RELEASECONTEXT.

Context\_Call MUST be initialized, as specified in section 2.2.2.2.

#### **3.2.5.3.4 Processing ReleaseContext Reply**

The response message MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.5 Sending IntroduceReader (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_INTRODUCEREADERA.

ContextAndTwoStringA\_Call MUST be initialized as specified in section 2.2.2.7 for a SCARD\_IOCTL\_INTRODUCEREADERA call.

#### **3.2.5.3.6 Processing IntroduceReader (ASCII) Reply**

The OutputBuffer MUST be decoded as a Long\_Return.

#### **3.2.5.3.7 Sending IntroduceReader (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_INTRODUCEREADERW.

ContextAndTwoStringW\_Call MUST be initialized, as specified in section 2.2.2.8, for a SCARD\_IOCTL\_INTRODUCEREADERW call.

#### **3.2.5.3.8 Processing IntroduceReader (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.9 Sending ForgetReader (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_FORGETREADER.

ContextAndStringA\_Call MUST be initialized, as specified in section 2.2.2.5, for a SCARD\_IOCTL\_FORGETREADER call.

#### **3.2.5.3.10 Processing ForgetReader (ASCII) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.11 Sending ForgetReader (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_FORGETREADERW.

ContextAndStringW\_Call MUST be initialized, as specified in section 2.2.2.6, for a SCARD\_IOCTL\_FORGETREADERW call.

### **3.2.5.3.12 Processing ForgetReader (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.13 Sending IntroduceReaderGroup (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_INTRODUCEREADERGROUPA.

ContextAndStringA\_Call MUST be initialized, as specified in section 2.2.2.5, for a SCARD\_IOCTL\_INTRODUCEREADERGROUPA call.

### **3.2.5.3.14 Processing IntroduceReaderGroup (ASCII) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.15 Sending IntroduceReaderGroup (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_INTRODUCEREADERGROUPW.

ContextAndStringW\_Call MUST be initialized, as specified in section 2.2.2.6, for a SCARD\_IOCTL\_INTRODUCEREADERGROUPW call.

### **3.2.5.3.16 Processing IntroduceReaderGroup (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.17 Sending ForgetReaderGroup (ASCII) Message 1**

IoControlCode MUST be set to SCARD\_IOCTL\_FORGETREADERGROUPA.

ContextAndStringA\_Call MUST be initialized, as specified in section 2.2.2.5, for a SCARD\_IOCTL\_FORGETREADERGROUPA call.

### **3.2.5.3.18 Processing ForgetReaderGroup (ASCII) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.19 Sending ForgetReaderGroup (ASCII) Message 2**

IoControlCode MUST be set to SCARD\_IOCTL\_FORGETREADERGROUPW.

ContextAndStringW\_Call MUST be initialized, as specified in section 2.2.2.6, for a SCARD\_IOCTL\_FORGETREADERGROUPW call.

### **3.2.5.3.20 Processing ForgetReaderGroup (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.21 Sending AddReaderToGroup (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_ADDREADERTOGROUPA.



ContextAndTwoStringA\_Call MUST be initialized, as specified in section 2.2.2.7, for a SCARD\_IOCTL\_ADDREADERTOGRROUPA call.

#### **3.2.5.3.22 Processing AddReaderToGroup (ASCII) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.23 Sending AddReaderToGroup (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_ADDREADERTOGRROUPW.

ContextAndTwoStringW\_Call MUST be initialized, as specified in section 2.2.2.8, for a SCARD\_IOCTL\_ADDREADERTOGRROUPW call.

#### **3.2.5.3.24 Processing AddReaderToGroup (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.25 Sending RemoveReaderFromGroup (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_REMOVEREADERFROMGRROUPA.

ContextAndTwoStringA\_Call MUST be initialized, as specified in section 2.2.2.7, for a SCARD\_IOCTL\_REMOVEREADERFROMGRROUPA call.

#### **3.2.5.3.26 Processing RemoveReaderFromGroup (ASCII) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.27 Sending RemoveReaderFromGroup (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_REMOVEREADERFROMGRROUPW.

ContextAndTwoStringW\_Call MUST be initialized, as specified in section 2.2.2.8, for a SCARD\_IOCTL\_REMOVEREADERFROMGRROUPW call.

#### **3.2.5.3.28 Processing RemoveReaderFromGroup (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.29 Sending ListReaderGroups (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LISTREADERGROUPSA.

ListReaderGroups\_Call MUST be initialized, as specified in section 2.2.2.3.

#### **3.2.5.3.30 Processing ListReaderGroups (ASCII) Reply**

The OutputBuffer MUST be decoded as ListReaderGroups\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.31 Sending ListReaderGroups (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LISTREADERGROUPSW.

ListReaderGroups\_Call MUST be initialized, as specified in section 2.2.2.3.

#### **3.2.5.3.32 Processing ListReaderGroups (Unicode) Reply**

The OutputBuffer MUST be decoded as ListReaderGroups\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.33 Sending ListReaders (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LISTREADERSA.

ListReaders\_Call MUST be initialized, as specified in section 2.2.2.4, for an ASCII call.

#### **3.2.5.3.34 Processing ListReadersReply (ASCII) Reply**

The OutputBuffer MUST be decoded as ListReaders\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.35 Sending ListReaders (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LISTREADERSW.

ListReaders\_Call MUST be initialized, as specified in section 2.2.2.4, for an Unicode call.

#### **3.2.5.3.36 Processing ListReadersReply (Unicode) Reply**

The OutputBuffer MUST be decoded as ListReaders\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.37 Sending LocateCards (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LOCATECARDSA.

LocateCardsA\_Call MUST be initialized as specified in section 2.2.2.9.

#### **3.2.5.3.38 Processing LocateCards (ASCII) Reply**

The OutputBuffer MUST be decoded as LocateCards\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.39 Sending LocateCards (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LOCATECARDSW.

LocateCardsW\_Call MUST be initialized, as specified in section 2.2.2.10.

#### **3.2.5.3.40 Processing LocateCards (Unicode) Reply**

The OutputBuffer MUST be decoded as LocateCards\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.41 Sending GetStatusChange (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_GETSTATUSCHANGEA.

GetStatusChangeA\_Call MUST be initialized, as specified in section 2.2.2.11.

#### **3.2.5.3.42 Processing GetStatusChange (ASCII) Reply**

The OutputBuffer MUST be decoded as GetStatusChange\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.43 Sending GetStatusChange (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_GETSTATUSCHANGEW.

GetStatusChangeW\_Call MUST be initialized, as specified in section 2.2.2.12.

#### **3.2.5.3.44 Processing GetStatusChange (Unicode) Reply**

The OutputBuffer MUST be decoded as GetStatusChange\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.45 Sending Cancel Message**

IoControlCode MUST be set to SCARD\_IOCTL\_CANCEL.

Context\_Call.Context MUST be initialized, as specified in section 2.2.2.2.

#### **3.2.5.3.46 Processing Cancel Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.47 Sending Connect (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_CONNECTA.

ConnectA\_Call MUST be initialized, as specified in section 2.2.2.13.

#### **3.2.5.3.48 Processing Connect (ASCII) Reply**

The OutputBuffer MUST be decoded as Connect\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.49 Sending Connect (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_CONNECTW.

ConnectW\_Call MUST be initialized, as specified in section 2.2.2.14.

#### **3.2.5.3.50 Processing Connect (Unicode) Reply**

The OutputBuffer MUST be decoded as Connect\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.51 Sending Reconnect Message**

IoControlCode MUST be set to SCARD\_IOCTL\_RECONNECT.

Reconnect\_Call MUST be initialized, as specified in section 2.2.2.15.

#### **3.2.5.3.52 Processing Reconnect Reply**

The OutputBuffer MUST be decoded as Reconnect\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.53 Sending Disconnect Message**

IoControlCode MUST be set to SCARD\_IOCTL\_DISCONNECT.

HCardAndDisposition\_Call MUST be initialized, as specified in section 2.2.2.16, for a SCARD\_IOCTL\_DISCONNECT call.

#### **3.2.5.3.54 Processing Disconnect Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.55 Sending Status (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_STATUSA.

Status\_Call MUST be initialized, as specified in section 2.2.2.18.

### **3.2.5.3.56 Processing Status (ASCII) Reply**

The OutputBuffer MUST be decoded as Status\_Return, as specified in [MS-RPCE] section 2.2.6, and interpreted as a SCARD\_IOCTL\_STATUSA return.

### **3.2.5.3.57 Sending Status (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_STATUSW.

Status\_Call MUST be initialized, as specified in section 2.2.2.18 .

### **3.2.5.3.58 Processing Status (Unicode) Reply**

The OutputBuffer MUST be decoded as Status\_Return, as specified in [MS-RPCE] section 2.2.6, and interpreted as a SCARD\_IOCTL\_STATUSW return.

### **3.2.5.3.59 Sending State Message**

IoControlCode MUST be set to SCARD\_IOCTL\_STATE.

State\_Call MUST be initialized, as specified in section 2.2.2.17, for a SCARD\_IOCTL\_STATE call.

### **3.2.5.3.60 Processing State Message Reply**

The OutputBuffer MUST be decoded as State\_Return, as specified in [MS-RPCE] section 2.2.6, and interpreted as a SCARD\_IOCTL\_STATE return.

### **3.2.5.3.61 Sending BeginTransaction Message**

IoControlCode MUST be set to SCARD\_IOCTL\_BEGINTRANSACTION.

HCardAndDisposition\_Call MUST be initialized, as specified in section 2.2.2.16, for a SCARD\_IOCTL\_BEGINTRANSACTION call.

### **3.2.5.3.62 Processing BeginTransaction Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.63 Sending EndTransaction Message**

IoControlCode MUST be set to SCARD\_IOCTL\_ENDTRANSACTION.

HCardAndDisposition\_Call MUST be initialized, as specified in section 2.2.2.16, for a SCARD\_IOCTL\_ENDTRANSACTION call.

### **3.2.5.3.64 Processing EndTransaction Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.65 Sending Transmit Message**

IoControlCode MUST be set to SCARD\_IOCTL\_TRANSMIT.

Transmit\_Call MUST be initialized as specified in section 2.2.2.19.

#### **3.2.5.3.66 Processing Transmit Reply**

The OutputBuffer MUST be decoded as Transmit\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.67 Sending Control Message**

IoControlCode MUST be set to SCARD\_IOCTL\_CONTROL.

Control\_Call MUST be initialized as specified in section 2.2.2.20.

#### **3.2.5.3.68 Processing Control Reply**

The OutputBuffer MUST be decoded as Control\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.69 Sending GetReaderCapabilities Message**

IoControlCode MUST be set to SCARD\_IOCTL\_GETATTRIB.

GetAttrib\_Call MUST be initialized as specified in section 2.2.2.21.

#### **3.2.5.3.70 Processing GetReaderCapabilities Reply**

The OutputBuffer MUST be decoded as GetAttrib\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.71 Sending SetReaderCapabilities Message**

IoControlCode MUST be set to SCARD\_IOCTL\_SETATTRIB.

SetAttrib\_Call MUST be initialized as specified in section 2.2.2.22.

#### **3.2.5.3.72 Processing SetReaderCapabilities Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.73 Sending WaitForResourceManager Message**

IoControlCode MUST be set to SCARD\_IOCTL\_ACCESSSTARTEDEVENT.

ScardAccessStartedEvent\_Call MUST be initialized as specified in section 2.2.2.30. This structure MUST NOT be encoded and MUST be sent as is.

#### **3.2.5.3.74 Processing WaitForResourceManager Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.75 Sending LocateCardsByATR (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LOCATECARDSBYATRA.

LocateCardsByATRA\_Call MUST be initialized as specified in section 2.2.2.23.

#### **3.2.5.3.76 Processing LocateCardsByATR (Unicode) Reply**

The OutputBuffer MUST be decoded as LocateCards\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.77 Processing LocateCardsByATR (ASCII) Reply**

The OutputBuffer MUST be decoded as LocateCards\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.78 Sending LocateCardsByATR (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_LOCATECARDSBYATRW.

LocateCardsByATRW\_Call MUST be initialized as specified in section 2.2.2.24.

#### **3.2.5.3.79 Sending ReadCache (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_READCACHEA.

ReadCacheA\_Call MUST be initialized as specified in section 2.2.2.25.

#### **3.2.5.3.80 Processing ReadCache (ASCII) Reply**

The OutputBuffer MUST be decoded as ReadCache\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.81 Sending ReadCache (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_READCACHEW.

ReadCacheW\_Call MUST be initialized as specified in section 2.2.2.26.

#### **3.2.5.3.82 Processing ReadCache (Unicode) Reply**

The OutputBuffer MUST be decoded as ReadCache\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.83 Sending WriteCache (ASCII) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_WRITECACHEA.

WriteCacheA\_Call MUST be initialized as specified in section 2.2.2.27.

#### **3.2.5.3.84 Processing WriteCache (ASCII) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.85 Sending WriteCache (Unicode) Message**

IoControlCode MUST be set to SCARD\_IOCTL\_WRITECACHEW.

WriteCacheW\_Call MUST be initialized as specified in section 2.2.2.28.

#### **3.2.5.3.86 Processing WriteCache (Unicode) Reply**

The OutputBuffer MUST be decoded as Long\_Return, as specified in [MS-RPCE] section 2.2.6.

#### **3.2.5.3.87 Sending GetTransmitCount Message**

IoControlCode MUST be set to SCARD\_IOCTL\_GETTRANSMITCOUNT.

GetTransmitCount\_Call MUST be initialized as specified in section 2.2.2.29.

#### **3.2.5.3.88 Processing GetTransmitCount Reply**

The OutputBuffer MUST be decoded as GetTransmitCount\_Return, as specified in [MS-RPCE] section 2.2.6.

### **3.2.5.3.89 Sending GetReaderIcon Message**

IoControlCode MUST be set to SCARD\_IOCTL\_GETREADERICON.

**GetReaderIcon\_Call** MUST be initialized as specified in section 2.2.2.31.

### **3.2.5.3.90 Processing GetReaderIcon Reply**

The **OutputBuffer** MUST be decoded as **GetReaderIcon\_Return**, as specified in section 2.2.3.14.

### **3.2.5.3.91 Sending GetDeviceTypeId Message**

IoControlCode MUST be set to SCARD\_IOCTL\_GETDEVICETYPEID.

**GetDeviceTypeId\_Call** MUST be initialized as specified in section 2.2.2.32.

### **3.2.5.3.92 Processing GetDeviceTypeId Reply**

The **OutputBuffer** MUST be decoded as **GetDeviceTypeId\_Return**, as specified in section 2.2.3.15.

## **3.2.6 Timer Events**

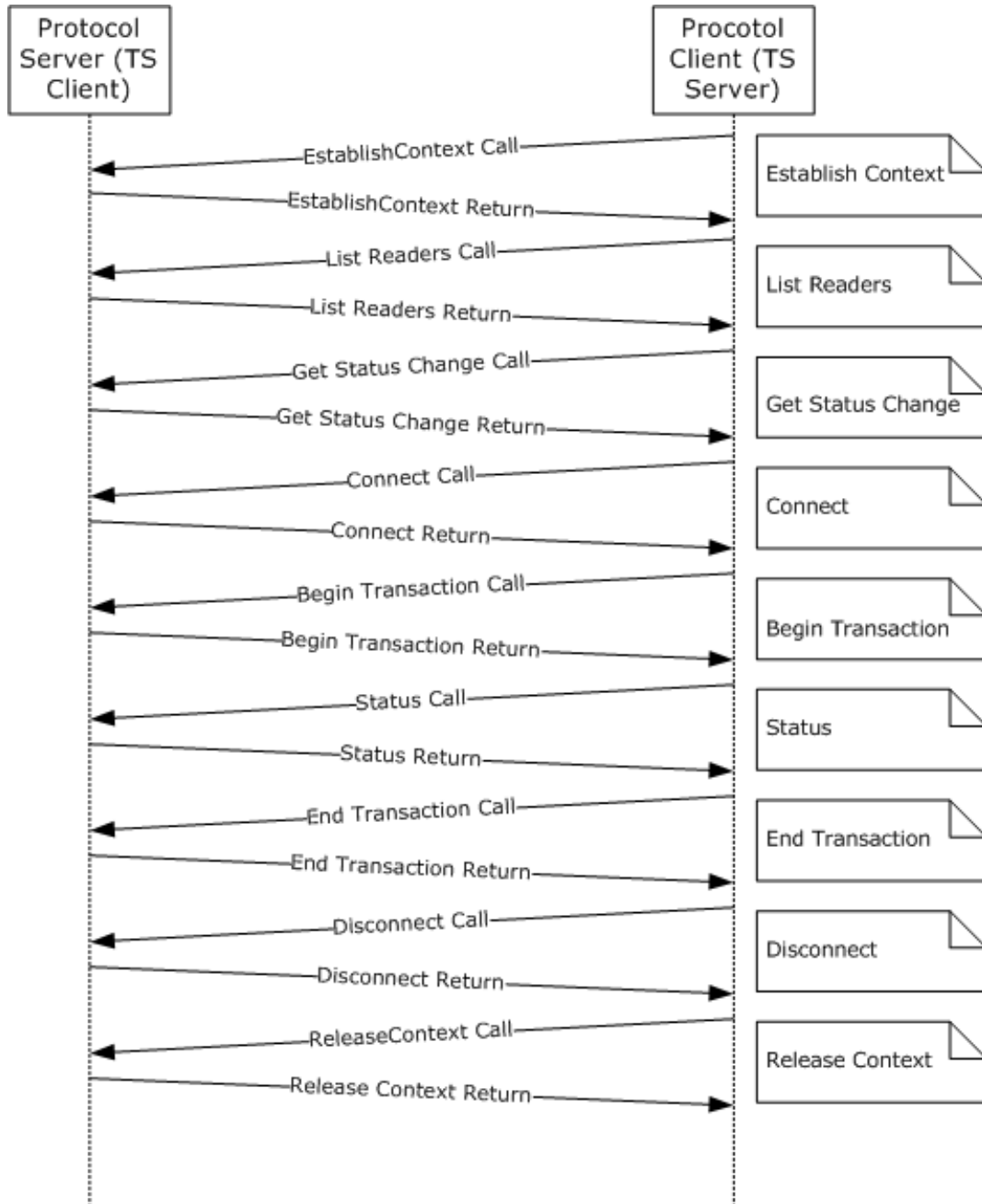
None.

## **3.2.7 Other Local Events**

None.

## 4 Protocol Examples

This example shows the messages sent to perform a simple querying of a card in the TS client machine. It assumes that a channel has already been set up on the between the TS client and the TS server. In addition, a PC/SC-compatible resource manager is running on the TS client and there exists a smart card reader with a smart card inserted. The following figure represents the program flow.



**Figure 4: Protocol flow**

This representation of the protocol flow is simplified in that there is only one application sending data over this protocol. In an actual implementation there could be multiple outstanding calls at any time.



All packets are constructed as specified in sections 3.2.5 and 3.2.5.3. The **Status** field refers to the **IoStatus** field as specified in [MS-RDPEFS] section 2.2.1.5. The **CompletionId** field is also specified in [MS-RDPEFS] section 2.2.1.5.

#### 4.1 Establish Context Call

```
IoControlCode= SCARD_IOCTL_ESTABLISHCONTEXT
CompletionId = 0
EstablishContext_Call {
  dwScope = SCARD_SCOPE_SYSTEM
}
```

The **CompletionId** field is specified in [MS-RDPEFS] section 2.2.1.4.

#### 4.2 Establish Context Return

```
CompletionId = 0
Status = 0
EstablishContext_Return {
  ReturnCode = 0
  Context = {cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
}
```

The **Status** field is specified as the **IoStatus** field in [MS-RDPEFS] section 2.2.1.5.

#### 4.3 List Readers Call

```
IoControlCode = SCARD_IOCTL_LISTREADERSW
CompletionId = 0
ListReaders_Call {
  Context = {cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
  cBytes = 44
  mszGroups = L"SCard$DefaultReaders\0\0"
  fmszReadersIsNULL = 0
  cchReaders = 0xFFFFFFFF
}
```

#### 4.4 List Readers Return

```
CompletionId = 0
Status = 0
ListReaders_Return {
  ReturnCode = 0
  cReaders = 66
  msz = L"Gemplus USB Smart Card Reader 0\0\0"
}
```

#### 4.5 Get Status Change Call

```
IoControlCode = SCARD_IOCTL_GETSTATUSCHANGEW
CompletionId = 0
GetStatusChangeW_Call {
  Context = {cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
  dwTimeOut = 0
  cReaders = 1
  rgReaderStates = {
  { szReader = L"Gemplus USB Smart Card Reader 0"
    Common = {
  dwCurrentState = SCARD_STATE_UNAWARE
```

```

dwEventState = 0
cbAtr = 0
pbAtr = {0} }
}
}
}

```

## 4.6 Get Status Change Return

```

Status = 0
CompletionId = 0
GetStatusChange_Return = {
  ReturnCode = 0
  cReaders = 1
  rgReaderStates = {
    dwCurrentState = SCARD_STATE_UNAWARE
    dwEventState = SCARD_STATE_CHANGED |
      SCARD_STATE_PRESENT | SCARD_STATE_INUSE
  }
  cbAtr = 9
  rgbAtr = {0x3b, 0x16, 0x94,0x41, 0x73, 0x74,0x72,0x69,
    0x64}
}
}

```

## 4.7 Connect Call

```

IoControlCode = SCARD_IOCTL_CONNECTW
CompletionId = 0
ConnectW_Call = {
  szReader = L"Gemplus USB Smart Card Reader 0"
  Common = {
    Context = { cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
    dwShareMode = SCARD_SHARE_SHARED
    dwPreferredProtocols = SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1
  }
}
}

```

## 4.8 Connect Return

```

CompletionId = 0
Status = 0
Connect_Return = {
  ReturnCode = 0
  hCard = {
    Context = { cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
    cbHandle = 4
    pbHandle = {0x00,0x00,0x01,0xea}{0x00,0x00,0x01,0xea}
    dwActiveProtocol = SCARD_PROTOCOL_T0
  }
}
}

```

## 4.9 Begin Transaction Call

```

IoControlCode = SCARD_IOCTL_BEGINTRANSACTION
CompletionId = 0
HCardAndDisposition_Call = {
  hCard = {
    Context = { cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
    cbHandle = 4
    pbHandle = {0x00,0x00,0x01,0xea}
    dwDisposition = 0
  }
}
}

```

## 4.10 Begin Transaction Return

```
CompletionId = 0
Status = 0
Long_Return = {
ReturnCode = 0
}
```

## 4.11 Status Call

```
IoControlCode = SCARD_IOCTL_STATUSW
CompletionId = 0
Status_Call = {
hCard = {
Context = { cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
cbHandle = 4
pbHandle = {0x00,0x00,0x01,0xea} }
fmszReaderNamesIsNULL = 0
cchReaderLen = 0xFFFFFFFF
cbAttrLen = 36
}
```

## 4.12 Status Return

```
CompletionId = 0
IoStatus = 0
Status_Return = {
ReturnCode = 0
cBytes = 66
mszReaderNames = L"Gemplus USB Smart Card Reader 0\0\0"
dwState = SCARD_SPECIFICMODE
dwProtocol = SCARD_PROTOCOL_T0
pbAtr = {0x3b, 0x16, 0x94,0x41, 0x73, 0x74,0x72,0x69,0x64}
cbAtr = 9
}
```

## 4.13 End Transaction Call

```
IoControlCode = SCARD_IOCTL_ENDTRANSACTION
CompletionId = 0
HCardAndDisposition_Call = {
hCard = {
Context = { cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
cbHandle = 4
pbHandle = {0x00,0x00,0x01,0xea}}
dwDisposition = SCARD_LEAVE_CARD
}
```

## 4.14 End Transaction Return

```
CompletionId = 0
Status = 0
Long_Return = {
ReturnCode = 0
}
```

## 4.15 Disconnect Call

```
IoControlCode = SCARD_IOCTL_DISCONNECT
```

```
CompletionId = 0
HCardAndDisposition_Call = {
hCard = {
Context = { cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
cbHandle = 4
pbHandle = {0x00,0x00,0x01,0xea}}
dwDisposition = SCARD_RESET_CARD
}
```

#### 4.16 Disconnect Return

```
CompletionId = 0
Status = 0
Long_Return = {
ReturnCode = 0
}
```

#### 4.17 Release Context Call

```
IoControlCode = SCARD_IOCTL_RELEASECONTEXT
CompletionId = 0
Context_Call = {
Context = {cbContext = 4, pbContext = {0x00,0x00,0x01,0xcd} }
}
```

#### 4.18 Release Context Return

```
CompletionId = 0
Status = 0
Long_Return = {
ReturnCode = 0
}
```

## **5 Security**

This protocol has no security aspects and relies on the underlying transport for any security.

### **5.1 Security Considerations for Implementers**

None.

### **5.2 Index of Security Parameters**

None.

## 6 Appendix A: Full IDL

For ease of implementation, the full Interface Definition Language (IDL) is provided below where ms-dtyp.idl is the IDL as specified in [MS-DTYP] section 5 and ms-dcom.idl is the IDL as specified in [MS-DCOM] section 6.

```
import "ms-dtyp.idl";
import "ms-dcom.idl";

[
    uuid(A35AF600-9CF4-11CD-A076-08002B2BD711),
    version(1.0),
    pointer_default(unique)
]
interface type_scard_pack
{
    //
    // Packing for calls that use the same params
    //
    typedef struct _REDIR_SCARDCONTEXT
    {
        [range(0, 16)]            unsigned long    cbContext;
        [unique] [size_is(cbContext)] byte        *pbContext;
    } REDIR_SCARDCONTEXT;

    typedef struct _REDIR_SCARDHANDLE
    {
        REDIR_SCARDCONTEXT        Context;
        [range(0, 16)]            unsigned long    cbHandle;
        [size_is(cbHandle)] byte    *pbHandle;
    } REDIR_SCARDHANDLE;

    typedef struct _long_Return
    {
        long                    ReturnCode;
    } long_Return;

    typedef struct _longAndMultiString_Return
    {
        long                    ReturnCode;
        [range(0, 65536)]        unsigned long    cBytes;
        [unique] [size_is(cBytes)] byte        *msz;
    } ListReaderGroups_Return, ListReaders_Return;

    typedef struct _Context_Call
    {
        REDIR_SCARDCONTEXT        Context;
    } Context_Call;

    typedef struct _ContextAndStringA_Call
    {
        REDIR_SCARDCONTEXT        Context;
        [string] const char *      sz;
    } ContextAndStringA_Call;

    typedef struct _ContextAndStringW_Call
    {
        REDIR_SCARDCONTEXT        Context;
        [string] const wchar_t *   sz;
    } ContextAndStringW_Call;

    typedef struct _ContextAndTwoStringA_Call
    {
        REDIR_SCARDCONTEXT        Context;
        [string] const char *      sz1;
    } ContextAndTwoStringA_Call;
}
```

```

    [string] const char *          sz2;
} ContextAndTwoStringA_Call;

typedef struct _ContextAndTwoStringW_Call
{
    REDIR_SCARDCONTEXT            Context;
    [string] const wchar_t *      sz1;
    [string] const wchar_t *      sz2;
} ContextAndTwoStringW_Call;

//
// Call specific packing
//
typedef struct _EstablishContext_Call
{
    unsigned long                 dwScope;
} EstablishContext_Call;

typedef struct _EstablishContext_Return
{
    long                          ReturnCode;
    REDIR_SCARDCONTEXT            Context;
} EstablishContext_Return;

typedef struct _ListReaderGroups_Call
{
    REDIR_SCARDCONTEXT            Context;
    long                          fmszGroupsIsNULL;
    unsigned long                 cchGroups;
} ListReaderGroups_Call;

typedef struct _ListReaders_Call
{
    REDIR_SCARDCONTEXT            Context;
    [range(0, 65536)] unsigned long cBytes;
    [unique] [size_is(cBytes)] const byte *mszGroups;
    long                          fmszReadersIsNULL;
    unsigned long                 cchReaders;
} ListReaders_Call;

typedef struct _ReaderState_Common_Call
{
    unsigned long                 dwCurrentState;
    unsigned long                 dwEventState;
    [range(0, 36)] unsigned long  cbAtr;
    byte                          rgbAtr[36];
} ReaderState_Common_Call;

typedef struct _ReaderStateA
{
    [string] const char *          szReader;
    ReaderState_Common_Call        Common;
} ReaderStateA;

typedef struct _ReaderStateW
{
    [string] const wchar_t *       szReader;
    ReaderState_Common_Call        Common;
} ReaderStateW;

typedef struct _ReaderState_Return
{
    unsigned long                 dwCurrentState;
    unsigned long                 dwEventState;
    [range(0, 36)] unsigned long  cbAtr;
    byte                          rgbAtr[36];
} ReaderState_Return;

```

```

typedef struct _GetStatusChangeA_Call
{
    REDIR_SCARDCONTEXT Context;
    unsigned long dwTimeOut;
    [range(0, 11)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateA *rgReaderStates;
} GetStatusChangeA_Call;

typedef struct _LocateCardsA_Call {
    REDIR_SCARDCONTEXT Context;
    [range(0, 65536)] unsigned long cBytes;
    [size_is(cBytes)] const byte * mszCards;
    [range(0, 10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateA * rgReaderStates;
} LocateCardsA_Call;

typedef struct _LocateCardsW_Call
{
    REDIR_SCARDCONTEXT Context;
    [range(0, 65536)] unsigned long cBytes;
    [size_is(cBytes)] const byte *mszCards;
    [range(0, 10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateW *rgReaderStates;
} LocateCardsW_Call;

typedef struct _LocateCards_ATRMask
{
    [range(0, 36)] unsigned long cbAtr;
    byte rgbAtr[36];
    byte rgbMask[36];
} LocateCards_ATRMask;

typedef struct _LocateCardsByATRA_Call
{
    REDIR_SCARDCONTEXT Context;
    [range(0, 1000)] unsigned long cAtrs;
    [size_is(cAtrs)] LocateCards_ATRMask *rgAtrMasks;
    [range(0, 10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateA *rgReaderStates;
} LocateCardsByATRA_Call;

typedef struct _LocateCardsByATRW_Call
{
    REDIR_SCARDCONTEXT Context;
    [range(0, 1000)] unsigned long cAtrs;
    [size_is(cAtrs)] LocateCards_ATRMask *rgAtrMasks;
    [range(0, 10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateW *rgReaderStates;
} LocateCardsByATRW_Call;

typedef struct _GetStatusChange_Return
{
    long ReturnCode;
    [range(0, 10)] unsigned long cReaders;
    [size_is(cReaders)] ReaderState_Return *rgReaderStates;
} LocateCards_Return, GetStatusChange_Return;

typedef struct _GetStatusChangeW_Call
{
    REDIR_SCARDCONTEXT Context;
    unsigned long dwTimeOut;
    [range(0, 11)] unsigned long cReaders;
    [size_is(cReaders)] ReaderStateW *rgReaderStates;
} GetStatusChangeW_Call;

```



```

typedef struct _Connect_Common
{
    REDIR_SCARDCONTEXT          Context;
    unsigned long               dwShareMode;
    unsigned long               dwPreferredProtocols;
} Connect_Common;

typedef struct _ConnectA_Call
{
    [string] const char *      szReader;
    Connect_Common             Common;
} ConnectA_Call;

typedef struct _ConnectW_Call
{
    [string] const wchar_t *   szReader;
    Connect_Common             Common;
} ConnectW_Call;

typedef struct _Connect_Return
{
    long                       ReturnCode;
    REDIR_SCARDHANDLE          hCard;
    unsigned long              dwActiveProtocol;
} Connect_Return;

typedef struct _Reconnect_Call
{
    REDIR_SCARDHANDLE          hCard;
    unsigned long              dwShareMode;
    unsigned long              dwPreferredProtocols;
    unsigned long              dwInitialization;
} Reconnect_Call;

typedef struct Reconnect_Return
{
    long                       ReturnCode;
    unsigned long              dwActiveProtocol;
} Reconnect_Return;

typedef struct _HCardAndDisposition_Call
{
    REDIR_SCARDHANDLE          hCard;
    unsigned long              dwDisposition;
} HCardAndDisposition_Call;

typedef struct _State_Call
{
    REDIR_SCARDHANDLE          hCard;
    long                       fpbAtrIsNull;
    unsigned long              cbAtrLen;
    // EDITOR'S NOTE: Can be 0xFFFFFFFF
} State_Call;

typedef struct _State_Return
{
    long                       ReturnCode;
    unsigned long              dwState;
    unsigned long              dwProtocol;
    [range(0, 36)]             cbAtrLen;
    [unique] [size_is(cbAtrLen)] byte *rgAtr;
} State_Return;

typedef struct _Status_Call
{
    REDIR_SCARDHANDLE          hCard;
    long                       fmszReaderNamesIsNull;
}

```

```

        unsigned long          cchReaderLen;
        unsigned long          cbAtrLen;
    } Status_Call;
typedef struct _Status_Return
{
        long          ReturnCode;
        [range(0, 65536)] unsigned long cBytes;
        [unique] [size_is(cBytes)] byte *mszReaderNames;
        unsigned long dwState;
        unsigned long dwProtocol;
        byte          pbAtr[32];
        [range(0, 32)] unsigned long cbAtrLen;
    } Status_Return;

typedef struct _SCardIO_Request
{
        unsigned long dwProtocol;
        [range(0, 1024)] unsigned long cbExtraBytes;
        [unique] [size_is(cbExtraBytes)] byte *pbExtraBytes;
    } SCardIO_Request;
typedef struct _Transmit_Call
{
        REDIR_SCARDHANDLE hCard;
        SCardIO_Request ioSendPci;
        [range(0, 66560)] unsigned long cbSendLength;
        [size_is(cbSendLength)] const byte *pbSendBuffer;
        [unique] SCardIO_Request *pioRecvPci;
        long fpbRecvBufferIsNULL;
        unsigned long cbRecvLength;
    } Transmit_Call;
typedef struct _Transmit_Return
{
        long          ReturnCode;
        [unique] SCardIO_Request *pioRecvPci;
        [range(0, 66560)] unsigned long cbRecvLength;
        [unique] [size_is(cbRecvLength)] byte *pbRecvBuffer;
    } Transmit_Return;

typedef struct _GetTransmitCount_Call
{
        REDIR_SCARDHANDLE hCard;
    } GetTransmitCount_Call;

typedef struct _GetTransmitCount_Return
{
        long          ReturnCode;
        unsigned long cTransmitCount;
    } GetTransmitCount_Return;

typedef struct _Control_Call
{
        REDIR_SCARDHANDLE hCard;
        unsigned long dwControlCode;
        [range(0, 66560)] unsigned long cbInBufferSize;
        [unique] [size_is(cbInBufferSize)] const byte *pvInBuffer;
        long fpvOutBufferIsNULL;
        unsigned long cbOutBufferSize;
    } Control_Call;

typedef struct _Control_Return
{
        long          ReturnCode;
        [range(0, 66560)] unsigned long cbOutBufferSize;
        [unique] [size_is(cbOutBufferSize)] byte *pvOutBuffer;
    } Control_Return;

typedef struct _GetAttrib_Call
{
        REDIR_SCARDHANDLE hCard;

```

```

        unsigned long          dwAttrId;
        long                fpbAttrIsNULL;
        unsigned long       cbAttrLen;
    } GetAttrib_Call;

typedef struct _GetAttrib_Return
{
    [range(0, 65536)]          long          ReturnCode;
    [unique] [size_is(cbAttrLen)] unsigned long cbAttrLen;
    [size_is(cbAttrLen)] byte    *pbAttr;
} GetAttrib_Return;

typedef struct _SetAttrib_Call
{
    REDIR_SCARDHANDLE         hCard;
    unsigned long            dwAttrId;
    [range(0, 65536)]        unsigned long cbAttrLen;
    [size_is(cbAttrLen)]    const byte    *pbAttr;
} SetAttrib_Call;

typedef struct _ReadCache_Common
{
    REDIR_SCARDCONTEXT       Context;
    UUID                     *CardIdentifier;
    unsigned long            FreshnessCounter;
    long                    fpbDataIsNULL;
    unsigned long            cbDataLen;
} ReadCache_Common;

typedef struct _ReadCacheA_Call
{
    [string] char *          szLookupName;
    ReadCache_Common        Common;
} ReadCacheA_Call;

typedef struct _ReadCacheW_Call
{
    [string] wchar_t *      szLookupName;
    ReadCache_Common        Common;
} ReadCacheW_Call;

typedef struct _ReadCache_Return
{
    [range(0, 65536)]          long          ReturnCode;
    [unique] [size_is(cbDataLen)] unsigned long cbDataLen;
    [size_is(cbDataLen)] byte    *pbData;
} ReadCache_Return;

typedef struct _WriteCache_Common
{
    REDIR_SCARDCONTEXT       Context;
    UUID                     *CardIdentifier;
    unsigned long            FreshnessCounter;
    [range(0, 65536)]        unsigned long cbDataLen;
    [unique] [size_is(cbDataLen)] byte    *pbData;
} WriteCache_Common;

typedef struct _WriteCacheA_Call
{
    [string] char *          szLookupName;
    WriteCache_Common        Common;
} WriteCacheA_Call;

typedef struct _WriteCacheW_Call
{
    [string] wchar_t *      szLookupName;
    WriteCache_Common        Common;
} WriteCacheW_Call;
}

```

## 7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Windows XP operating system
- Windows Server 2003 operating system
- 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
- Windows 10 operating system
- Windows Server 2016 operating system
- Windows Server operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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 1.7: The Windows XP and Windows Server 2003 versions always use SCREDIR\_VERSION\_XP. Windows Vista and Windows Server 2008 are always SCREDIR\_VERSION\_LONGHORN. All other versions use SCREDIR\_VERSION\_WINDOWS\_8.

<2> Section 3.1.4: Windows XP and Windows Server 2003 implement function numbers 5 through 58. Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 implement function numbers 5 through 64. All other versions implement 5 through 66.

## 8 Change Tracking

~~This section identifies **No table** of changes that were made to this is available. The document is either new or has had no changes since theits last release. Changes are classified as Major, Minor, or None.~~

~~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.~~
- ~~• A document revision that captures changes to protocol functionality.~~

~~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 **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.~~

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

<b>Section</b>	<b>Description</b>	<b>Revision class</b>
<del>7 Appendix B: Product Behavior</del>	<del>Added Windows Server operating system to the list of applicable products.</del>	<del>Major</del>

## 9 Index

### A

- Abstract data model
  - client 62
  - server 49
- Access\_Mode\_Flags packet 43
- Applicability 16

### B

- Begin transaction call example 74
- Begin transaction return example 75

### C

- Capability negotiation 16
- Card\_Reader\_State packet 41
- Change tracking 85
- Client
  - abstract data model 62
  - higher-layer triggered events 62
  - initialization 62
  - local events 71
  - message processing 62
  - Processing Incoming Replies method 62
  - Sending Outgoing Messages method 62
  - sequencing rules 62
  - structures (section 2.2.1 18, section 2.2.3 36)
  - timer events 71
  - timers 62
- Common data types 18
- Connect call example 74
- Connect return example 74
- Connect\_Common structure 19
- Connect\_Return structure 38
- ConnectA\_Call structure 28
- ConnectW\_Call structure 28
- Context\_Call structure 22
- ContextAndStringA\_Call structure 24
- ContextAndStringW\_Call structure 24
- ContextAndTwoStringA\_Call structure 25
- ContextAndTwoStringW\_Call structure 26
- Control\_Call structure 31
- Control\_Return structure 38

### D

- Data model - abstract
  - client 62
  - server 49
- Data types 18
  - common - overview 18
- Disconnect call example 75
- Disconnect return example 76

### E

- End transaction call example 75
- End transaction return example 75
- Establish context call example 73
- Establish context return example 73

EstablishContext\_Call structure 22  
EstablishContext\_Return structure 36  
Events  
  local - client 71  
  local - server 61  
  timer - client 71  
  timer - server 61  
Examples  
  begin transaction call 74  
  begin transaction call example 74  
  begin transaction return 75  
  begin transaction return example 75  
  connect call 74  
  connect call example 74  
  connect return 74  
  connect return example 74  
  disconnect call 75  
  disconnect call example 75  
  disconnect return 76  
  disconnect return example 76  
  end transaction call 75  
  end transaction call example 75  
  end transaction return 75  
  end transaction return example 75  
  establish context call 73  
  establish context call example 73  
  establish context return 73  
  establish context return example 73  
  get status change call 73  
  get status change call example 73  
  get status change return 74  
  get status change return example 74  
  list reader call example 73  
  list reader return example 73  
  list readers call 73  
  list readers return 73  
  overview 72  
  release context call 76  
  release context call example 76  
  release context return 76  
  release context return example 76  
  status call 75  
  status call example 75  
  status return 75  
  status return example 75

## **F**

Fields - vendor-extensible 16  
Full IDL 78

## **G**

Get status change call example 73  
Get status change return example 74  
GetAttrib\_Call structure 32  
GetAttrib\_Return structure 40  
GetDeviceTypeId\_Call structure 36  
GetDeviceTypeId\_Return structure 41  
GetReaderIcon\_Call structure 35  
GetReaderIcon\_Return structure 41  
GetStatusChange\_Return 37  
GetStatusChangeA\_Call structure 27  
GetStatusChangeW\_Call structure 27  
GetTransmitCount\_Call structure 35

GetTransmitCount\_Return structure 41  
Glossary 10

## H

HCardAndDisposition\_Call structure 29  
Higher-layer triggered events - client 62

## I

IDL 78  
Implementer - security considerations 77  
Implementers - security considerations 77  
Index of security parameters 77  
Informative references 13  
Initialization  
    client 62  
    server 49  
Introduction 10

## L

List reader call example 73  
List reader return example 73  
List readers call example 73  
List readers return example 73  
ListReaderGroups\_Call structure 23  
ListReaderGroups\_Return structure 37  
ListReaders\_Call structure 23  
ListReaders\_Return 37  
Local events  
    client 71  
    server 61  
LocateCards\_ATRMask structure 19  
LocateCards\_Return structure 37  
LocateCardsA\_Call structure 26  
LocateCardsByATRA\_Call structure 33  
LocateCardsByATRW\_Call structure 34  
LocateCardsW\_Call structure 27  
Long\_Return structure 37

## M

Message processing  
    client 62  
    server 49  
Messages  
    common data types 18  
    names 63  
    overview 18  
    processing incoming replies 62  
    sending outgoing messages 62  
    transport 18  
Methods  
    Processing Incoming Replies 62  
    SCARD\_IOCTL\_ACCESSSTARTEDEVENT (IOCTL 0x000900E0) 53  
    SCARD\_IOCTL\_ADDREADERTOGROUPA (IOCTL 0x00090070) 56  
    SCARD\_IOCTL\_ADDREADERTOGROUPW (IOCTL 0x00090074) 56  
    SCARD\_IOCTL\_BEGINTRANSACTION (IOCTL 0x000900BC) 58  
    SCARD\_IOCTL\_CANCEL (IOCTL 0x000900A8) 58  
    SCARD\_IOCTL\_CONNECTA (IOCTL 0x000900AC) 58  
    SCARD\_IOCTL\_CONNECTW (IOCTL 0x000900B0) 58  
    SCARD\_IOCTL\_CONTROL (IOCTL 0x000900D4) 59  
    SCARD\_IOCTL\_DISCONNECT (IOCTL 0x000900B8) 58  
    SCARD\_IOCTL\_ENDTRANSACTION (IOCTL 0x000900C0) 58



SCARD\_IOCTL\_ESTABLISHCONTEXT (IOCTL 0x00090014) 53  
SCARD\_IOCTL\_FORGETREADER (IOCTL 0x00090068) 55  
SCARD\_IOCTL\_FORGETREADERGROUPA (IOCTL 0x00090058) 55  
SCARD\_IOCTL\_FORGETREADERGROUPW (IOCTL 0x0009005C) 55  
SCARD\_IOCTL\_FORGETREADERW (IOCTL 0x0009006C) 56  
SCARD\_IOCTL\_GETATTRIB (IOCTL 0x000900D8) 59  
SCARD\_IOCTL\_GETDEVICETYPEID (IOCTL 0x00090108) 61  
SCARD\_IOCTL\_GETREADERICON (IOCTL 0x00090104) 61  
SCARD\_IOCTL\_GETSTATUSCHANGEA (IOCTL 0x000900A0) 57  
SCARD\_IOCTL\_GETSTATUSCHANGEW (IOCTL 0x000900A4) 57  
SCARD\_IOCTL\_GETTRANSMITCOUNT (IOCTL 0x00090100) 60  
SCARD\_IOCTL\_INTRODUCEREADER (IOCTL 0x00090060) 55  
SCARD\_IOCTL\_INTRODUCEREADERGROUPA (IOCTL 0x00090050) 54  
SCARD\_IOCTL\_INTRODUCEREADERGROUPW (IOCTL 0x00090054) 55  
SCARD\_IOCTL\_INTRODUCEREADERW (IOCTL 0x00090064) 55  
SCARD\_IOCTL\_ISVALIDCONTEXT (IOCTL 0x0009001C) 53  
SCARD\_IOCTL\_LISTREADERGROUPSA (IOCTL 0x00090020) 54  
SCARD\_IOCTL\_LISTREADERGROUPSW (IOCTL 0x00090024) 54  
SCARD\_IOCTL\_LISTREADERSA (IOCTL 0x00090028) 54  
SCARD\_IOCTL\_LISTREADERSW (IOCTL 0x0009002C) 54  
SCARD\_IOCTL\_LOCATECARDSA (IOCTL 0x00090098) 56  
SCARD\_IOCTL\_LOCATECARDSBYATRA (IOCTL 0x000900E8) 57  
SCARD\_IOCTL\_LOCATECARDSBYATR (IOCTL 0x000900EC) 57  
SCARD\_IOCTL\_LOCATECARDSW (IOCTL 0x0009009C) 57  
SCARD\_IOCTL\_READCACHEA (IOCTL 0x000900F0) 60  
SCARD\_IOCTL\_READCACHEW (IOCTL 0x000900F4) 60  
SCARD\_IOCTL\_RECONNECT (IOCTL 0x000900B4) 59  
SCARD\_IOCTL\_RELEASECONTEXT (IOCTL 0x00090018) 53  
SCARD\_IOCTL\_RELEASESTARTEVENT 61  
SCARD\_IOCTL\_REMOVEREADERFROMGROUPA (IOCTL 0x00090078) 56  
SCARD\_IOCTL\_REMOVEREADERFROMGROUPW (IOCTL 0x0009007C) 56  
SCARD\_IOCTL\_SETATTRIB (IOCTL 0x000900DC) 60  
SCARD\_IOCTL\_STATE (IOCTL 0x000900C4) 60  
SCARD\_IOCTL\_STATUSA (IOCTL 0x000900C8) 59  
SCARD\_IOCTL\_STATUSW (IOCTL 0x000900CC) 59  
SCARD\_IOCTL\_TRANSMIT (IOCTL 0x000900D0) 59  
SCARD\_IOCTL\_WRITECACHEA (IOCTL 0x000900F8) 61  
SCARD\_IOCTL\_WRITECACHEW (IOCTL 0x000900FC) 61  
Sending Outgoing Messages 62

## **N**

Normative references 12

## **O**

Outgoing messages - sending 62  
Overview (synopsis) 13

## **P**

Parameters - security 77  
Parameters - security index 77  
Preconditions 15  
Prerequisites 15  
Processing Incoming Replies method 62  
Product behavior 84  
Protocol Details  
    overview 49  
Protocol\_Identifier packet 42

## **R**

ReadCache\_Common structure 21  
ReadCache\_Return structure 36

- ReadCacheA\_Call structure 34
- ReadCacheW\_Call structure 34
- Reader\_State packet 43
- ReaderState\_Common\_Call structure 19
- ReaderState\_Return structure 21
- ReaderStateA structure 20
- ReaderStateW structure 20
- Reconnect\_Call structure 28
- Reconnect\_Return structure 38
- REDIR\_SCARDCONTEXT structure 18
- REDIR\_SCARDHANDLE structure 18
- References 12
  - informative 13
  - normative 12
- Relationship to other protocols 15
- Release context call example 76
- Release context return example 76
- Replies - processing 62
- Return\_Code packet 45

## S

- SCARD\_IOCTL\_ACCESSSTARTEDEVENT (IOCTL 0x000900E0) method 53
- SCARD\_IOCTL\_ADDREADERTOGROUPA (IOCTL 0x00090070) method 56
- SCARD\_IOCTL\_ADDREADERTOGROUPW (IOCTL 0x00090074) method 56
- SCARD\_IOCTL\_BEGINTRANSACTION (IOCTL 0x000900BC) method 58
- SCARD\_IOCTL\_CANCEL (IOCTL 0x000900A8) method 58
- SCARD\_IOCTL\_CONNECTA (IOCTL 0x000900AC) method 58
- SCARD\_IOCTL\_CONNECTW (IOCTL 0x000900B0) method 58
- SCARD\_IOCTL\_CONTROL (IOCTL 0x000900D4) method 59
- SCARD\_IOCTL\_DISCONNECT (IOCTL 0x000900B8) method 58
- SCARD\_IOCTL\_ENDTRANSACTION (IOCTL 0x000900C0) method 58
- SCARD\_IOCTL\_ESTABLISHCONTEXT (IOCTL 0x00090014) method 53
- SCARD\_IOCTL\_FORGETREADERA (IOCTL 0x00090068) method 55
- SCARD\_IOCTL\_FORGETREADERGROUPA (IOCTL 0x00090058) method 55
- SCARD\_IOCTL\_FORGETREADERGROUPW (IOCTL 0x0009005C) method 55
- SCARD\_IOCTL\_FORGETREADERW (IOCTL 0x0009006C) method 56
- SCARD\_IOCTL\_GETATTRIB (IOCTL 0x000900D8) method 59
- SCARD\_IOCTL\_GETDEVICETYPEID (IOCTL 0x00090108) method 61
- SCARD\_IOCTL\_GETREADERICON (IOCTL 0x00090104) method 61
- SCARD\_IOCTL\_GETSTATUSCHANGEA (IOCTL 0x000900A0) method 57
- SCARD\_IOCTL\_GETSTATUSCHANGEW (IOCTL 0x000900A4) method 57
- SCARD\_IOCTL\_GETTRANSMITCOUNT (IOCTL 0x00090100) method 60
- SCARD\_IOCTL INTRODUCEREADERA (IOCTL 0x00090060) method 55
- SCARD\_IOCTL INTRODUCEREADERGROUPA (IOCTL 0x00090050) method 54
- SCARD\_IOCTL INTRODUCEREADERGROUPW (IOCTL 0x00090054) method 55
- SCARD\_IOCTL INTRODUCEREADERW (IOCTL 0x00090064) method 55
- SCARD\_IOCTL\_ISVALIDCONTEXT (IOCTL 0x0009001C) method 53
- SCARD\_IOCTL\_LISTREADERGROUPSA (IOCTL 0x00090020) method 54
- SCARD\_IOCTL\_LISTREADERGROUPSW (IOCTL 0x00090024) method 54
- SCARD\_IOCTL\_LISTREADERSA (IOCTL 0x00090028) method 54
- SCARD\_IOCTL\_LISTREADERSW (IOCTL 0x0009002C) method 54
- SCARD\_IOCTL\_LOCATECARDSA (IOCTL 0x00090098) method 56
- SCARD\_IOCTL\_LOCATECARDSBYATRA (IOCTL 0x000900E8) method 57
- SCARD\_IOCTL\_LOCATECARDSBYATRW (IOCTL 0x000900EC) method 57
- SCARD\_IOCTL\_LOCATECARDSW (IOCTL 0x0009009C) method 57
- SCARD\_IOCTL\_READCACHEA (IOCTL 0x000900F0) method 60
- SCARD\_IOCTL\_READCACHEW (IOCTL 0x000900F4) method 60
- SCARD\_IOCTL\_RECONNECT (IOCTL 0x000900B4) method 59
- SCARD\_IOCTL\_RELEASECONTEXT (IOCTL 0x00090018) method 53
- SCARD\_IOCTL\_RELEASESTARTEDEVENT method 61
- SCARD\_IOCTL\_REMOVEREADERFROMGROUPA (IOCTL 0x00090078) method 56
- SCARD\_IOCTL\_REMOVEREADERFROMGROUPW (IOCTL 0x0009007C) method 56
- SCARD\_IOCTL\_SETATTRIB (IOCTL 0x000900DC) method 60
- SCARD\_IOCTL\_STATE (IOCTL 0x000900C4) method 60

SCARD\_IOCTL\_STATUSA (IOCTL 0x000900C8) method 59  
 SCARD\_IOCTL\_STATUSW (IOCTL 0x000900CC) method 59  
 SCARD\_IOCTL\_TRANSMIT (IOCTL 0x000900D0) method 59  
 SCARD\_IOCTL\_WRITECACHEA (IOCTL 0x000900F8) method 61  
 SCARD\_IOCTL\_WRITECACHEW (IOCTL 0x000900FC) method 61  
 ScardAccessStartedEvent\_Call packet 35  
 SCardIO\_Request structure 20  
 Security 77  
   implementer considerations 77  
   parameter index 77  
 Sending Outgoing Messages method 62  
 Sequencing rules  
   client 62  
   server 49  
 Server  
   abstract data model 49  
   initialization 49  
   local events 61  
   message processing 49  
   SCARD\_IOCTL\_ACCESSSTARTEDEVENT (IOCTL 0x000900E0) method 53  
   SCARD\_IOCTL\_ADDREADERTOGROUPA (IOCTL 0x00090070) method 56  
   SCARD\_IOCTL\_ADDREADERTOGROUPW (IOCTL 0x00090074) method 56  
   SCARD\_IOCTL\_BEGINTRANSACTION (IOCTL 0x000900BC) method 58  
   SCARD\_IOCTL\_CANCEL (IOCTL 0x000900A8) method 58  
   SCARD\_IOCTL\_CONNECTA (IOCTL 0x000900AC) method 58  
   SCARD\_IOCTL\_CONNECTW (IOCTL 0x000900B0) method 58  
   SCARD\_IOCTL\_CONTROL (IOCTL 0x000900D4) method 59  
   SCARD\_IOCTL\_DISCONNECT (IOCTL 0x000900B8) method 58  
   SCARD\_IOCTL\_ENDTRANSACTION (IOCTL 0x000900C0) method 58  
   SCARD\_IOCTL\_ESTABLISHCONTEXT (IOCTL 0x00090014) method 53  
   SCARD\_IOCTL\_FORGETREADERA (IOCTL 0x00090068) method 55  
   SCARD\_IOCTL\_FORGETREADERGROUPA (IOCTL 0x00090058) method 55  
   SCARD\_IOCTL\_FORGETREADERGROUPW (IOCTL 0x0009005C) method 55  
   SCARD\_IOCTL\_FORGETREADERW (IOCTL 0x0009006C) method 56  
   SCARD\_IOCTL\_GETATTRIB (IOCTL 0x000900D8) method 59  
   SCARD\_IOCTL\_GETDEVICETYPEID (IOCTL 0x00090108) method 61  
   SCARD\_IOCTL\_GETREADERICON (IOCTL 0x00090104) method 61  
   SCARD\_IOCTL\_GETSTATUSCHANGEA (IOCTL 0x000900A0) method 57  
   SCARD\_IOCTL\_GETSTATUSCHANGEW (IOCTL 0x000900A4) method 57  
   SCARD\_IOCTL\_GETTRANSMITCOUNT (IOCTL 0x00090100) method 60  
   SCARD\_IOCTL INTRODUCEREADERA (IOCTL 0x00090060) method 55  
   SCARD\_IOCTL INTRODUCEREADERGROUPA (IOCTL 0x00090050) method 54  
   SCARD\_IOCTL INTRODUCEREADERGROUPW (IOCTL 0x00090054) method 55  
   SCARD\_IOCTL INTRODUCEREADERW (IOCTL 0x00090064) method 55  
   SCARD\_IOCTL\_ISVALIDCONTEXT (IOCTL 0x0009001C) method 53  
   SCARD\_IOCTL\_LISTREADERGROUPSA (IOCTL 0x00090020) method 54  
   SCARD\_IOCTL\_LISTREADERGROUPSW (IOCTL 0x00090024) method 54  
   SCARD\_IOCTL\_LISTREADERSA (IOCTL 0x00090028) method 54  
   SCARD\_IOCTL\_LISTREADERSW (IOCTL 0x0009002C) method 54  
   SCARD\_IOCTL\_LOCATECARDSA (IOCTL 0x00090098) method 56  
   SCARD\_IOCTL\_LOCATECARDSBYATRA (IOCTL 0x000900E8) method 57  
   SCARD\_IOCTL\_LOCATECARDSBYATRW (IOCTL 0x000900EC) method 57  
   SCARD\_IOCTL\_LOCATECARDSW (IOCTL 0x0009009C) method 57  
   SCARD\_IOCTL\_READCACHEA (IOCTL 0x000900F0) method 60  
   SCARD\_IOCTL\_READCACHEW (IOCTL 0x000900F4) method 60  
   SCARD\_IOCTL\_RECONNECT (IOCTL 0x000900B4) method 59  
   SCARD\_IOCTL\_RELEASECONTEXT (IOCTL 0x00090018) method 53  
   SCARD\_IOCTL\_RELEASESTARTEDEVENT method 61  
   SCARD\_IOCTL\_REMOVEREADERFROMGROUPA (IOCTL 0x00090078) method 56  
   SCARD\_IOCTL\_REMOVEREADERFROMGROUPW (IOCTL 0x0009007C) method 56  
   SCARD\_IOCTL\_SETATTRIB (IOCTL 0x000900DC) method 60  
   SCARD\_IOCTL\_STATE (IOCTL 0x000900C4) method 60  
   SCARD\_IOCTL\_STATUSA (IOCTL 0x000900C8) method 59  
   SCARD\_IOCTL\_STATUSW (IOCTL 0x000900CC) method 59  
   SCARD\_IOCTL\_TRANSMIT (IOCTL 0x000900D0) method 59

- SCARD\_IOCTL\_WRITECACHEA (IOCTL 0x000900F8) method 61
- SCARD\_IOCTL\_WRITECACHEW (IOCTL 0x000900FC) method 61
- sequencing rules 49
- structures (section 2.2.1 18, section 2.2.2 22)
- timer events 61
- timers 49
- SetAttrib\_Call structure 33
- Standards assignments 16
- State\_Call structure 30
- State\_Return structure 39
- Status call example 75
- Status return example 75
- Status\_Call structure 30
- Status\_Return structure 39
- Structures
  - client (section 2.2.1 18, section 2.2.3 36)
  - server (section 2.2.1 18, section 2.2.2 22)

## **T**

- Timer events
  - client 71
  - server 61
- Timers
  - client 62
  - server 49
- Tracking changes 85
- Transmit\_Call structure 31
- Transmit\_Return structure 40
- Transport 18
- Transport - message 18
- Triggered events - higher-layer - client 62

## **V**

- Vendor-extensible fields 16
- Versioning 16

## **W**

- WriteCache\_Common structure 21
- WriteCacheA\_Call structure 34
- WriteCacheW\_Call structure 35