International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.128

(06/2008)

SERIES T: TERMINALS FOR TELEMATIC SERVICES

Data protocols for multimedia conferencing

Multipoint application sharing

Recommendation ITU-T T.128

ITU-T T-SERIES RECOMMENDATIONS

TERMINALS FOR TELEMATIC SERVICES

Facsimile – Framework	T.0–T.19
Still-image compression – Test charts	T.20–T.29
Facsimile – Group 3 protocols	T.30–T.39
Colour representation	T.40–T.49
Character coding	T.50–T.59
Facsimile – Group 4 protocols	T.60–T.69
Telematic services – Framework	T.70–T.79
Still-image compression – JPEG-1, Bi-level and JBIG	T.80–T.89
Telematic services – ISDN Terminals and protocols	T.90–T.99
Videotext – Framework	T.100–T.109
Data protocols for multimedia conferencing	**T.120–T.149**
Telewriting	T.150–T.159
Multimedia and hypermedia framework	T.170–T.189
Cooperative document handling	T.190–T.199
Telematic services – Interworking	T.300–T.399
Open document architecture	T.400–T.429
Document transfer and manipulation	T.430–T.449
Document application profile	T.500–T.509
Communication application profile	T.510–T.559
Telematic services – Equipment characteristics	T.560–T.649
Still-image compression – JPEG 2000	T.800–T.849
Still-image compression – JPEG-1 extensions	T.850–T.899

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T T.128

Multipoint application sharing

Summary

Recommendation ITU-T T.128 defines a protocol that supports multipoint application sharing.

The T.128 protocol supports multipoint computer application sharing by allowing a view onto a computer application executing at one site to be advertised within a session to other sites. Each site can, under specified conditions, take control of the shared computer application by sending remote keyboard and pointing device information. This style of application sharing does not require and does not make provision for synchronizing multiple instances of the same computer application running at multiple sites. Instead, it enables remote viewing and control of a single application instance to provide the illusion that the application is running locally.

This Recommendation uses services provided by Recommendations ITU-T T.122 (MCS) and T.124 (GCC).

This revised version of Recommendation ITU-T T.128 introduces a number of modifications to legacy mode ASN.1 definition of the application sharing protocol.

Source

Recommendation ITU-T T.128 was approved on 13 June 2008 by ITU-T Study Group 16 (2005-2008) under Recommendation ITU-T A.8 procedure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

CONTENTS

Recommendation ITU-T T.128

Multipoint application sharing

1 Scope

This Recommendation defines a protocol that supports multipoint application sharing. It uses services provided by [ITU-T T.122] (MCS) and [ITU-T T.124] (GCC).

The details of communication with the input and output devices and the user interfaces on the host terminal are considered out of the scope of this Recommendation and are left to the discretion of the implementer. Therefore, this Recommendation makes no assumption that these input and output devices and user interfaces are of any specific architecture.

Figure 1 presents an overview of the scope of this Recommendation and its relationship to the other elements of the T.120 framework within a single node.

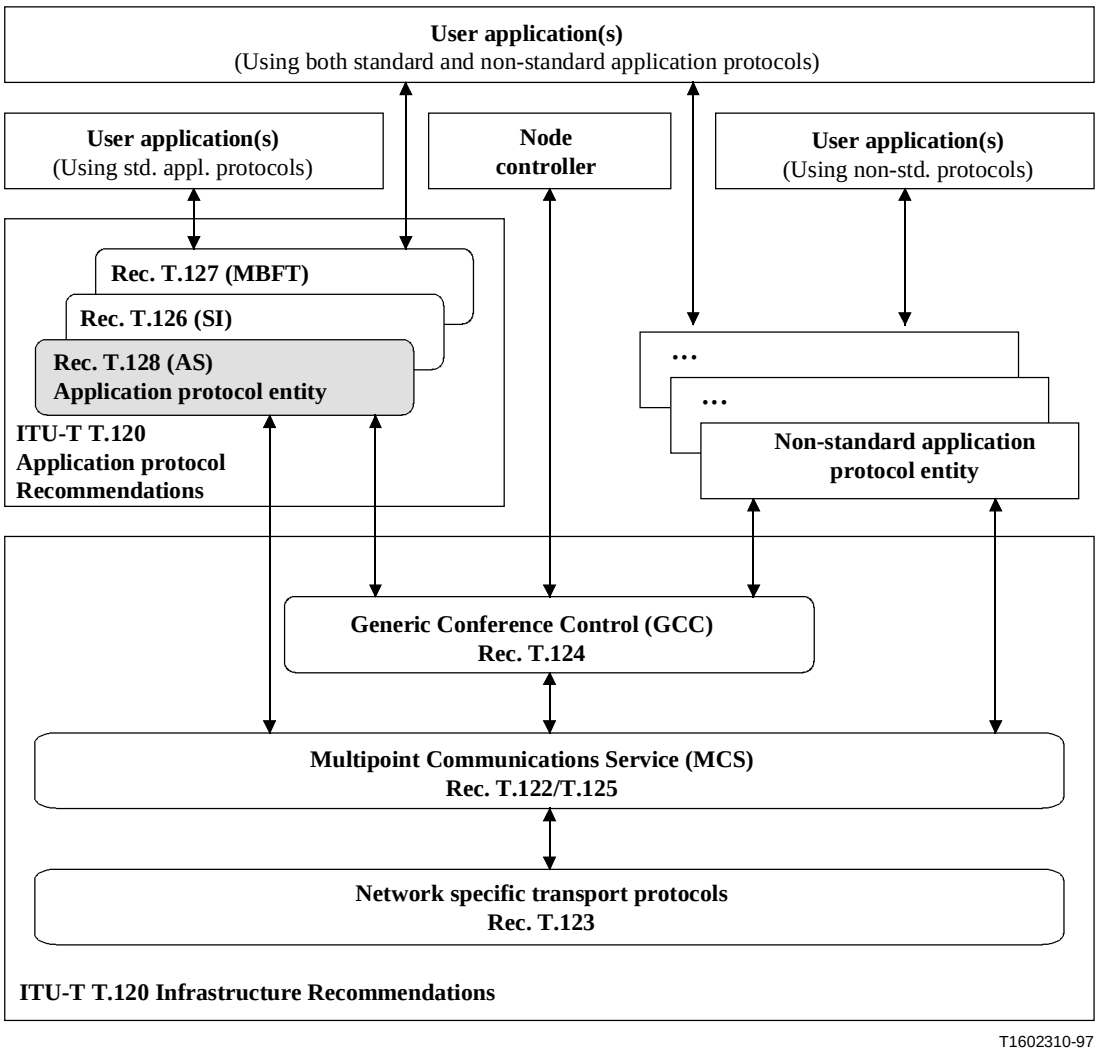


Figure 1 – Scope of this Recommendation

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T F.710] Recommendation ITU-T F.710 (1991), *General principles for audiographic conference service.*

[ITU-T H.221] Recommendation ITU-T H.221 (1997), *Frame structure for a 64 to 1920 kbit/s channel in audiovisual teleservices.*

[ITU-T T.35] Recommendation ITU-T T.35 (1991), *Procedure for the allocation of CCITT defined codes for non-standard facilities.*

[ITU-T T.50] Recommendation ITU-T T.50 (1992), *International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) – Information technology – 7-bit coded character set for information interchange.*

[ITU-T T.120] Recommendation ITU-T T.120 (2007), *Data protocols for multimedia conferencing.*

[ITU-T T.121] Recommendation ITU-T T.121 (1996), *Generic application template.*

[ITU-T T.122] Recommendation ITU-T T.122 (1993), *Multipoint communication service for audiographics and audiovisual conferencing service definition.*

[ITU-T T.123] Recommendation ITU-T T.123 (1996), *Network-specific data protocol stacks for multimedia conferencing.*

[ITU-T T.124] Recommendation ITU-T T.124 (1995), *Generic Conference Control.*

[ITU-T T.125] Recommendation ITU-T T.125 (1994), *Multipoint communication service protocol specification.*

[ITU-T V.42 *bis*] Recommendation ITU-T V.42 *bis* (1990), *Data compression procedures for data circuit-terminating equipment (DCE) using error correcting procedures.*

[ITU-T X.509] Recommendation ITU-T X.509 (1997) | ISO/IEC 9594-8:1998, *Information technology – Open Systems Interconnection – The Directory: Authentication framework.*

[ITU-T X.680] Recommendation ITU-T X.680 (1994) | ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1) – Specification of basic notation.*

[ITU-T X.691] Recommendation ITU-T X.691 (1997) | ISO/IEC 8825-2:1998, *Information technology – ASN.1 encoding rules – Specification of Packed Encoding Rules (PER).*

[ISO/IEC 8859-1] ISO/IEC 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1.* <http://iso.org/iso/iso_catalogue_tc/catalogue_details.htm?csnumber=28245>

[ISO/IEC 10646] ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS).* <http://iso.org/iso/iso_catalogue_tc/catalogue_details.htm?csnumber=39921>

3 Definitions

This Recommendation defines the following terms:

3.1 application sharing (AS): A process whereby two or more terminals cooperate to share the output of applications running on one or more terminals to the other terminals and to provide input to the applications.

3.2 application sharing conference entity (ASCE): An application protocol entity that interacts with a user application above and with the local multipoint communication service (MCS) and local generic conference control (GCC) providers below to implement application sharing. Data are exchanged between peer ASCEs using application sharing protocol data units (ASPDUs).

3.3 bitmap: A rectangular area described by a two-dimensional array of pixels. These pixels can be coded using a variety of encoding methods.

3.4 ColorTable: A finite set of colours defined by at least three linearly-independent colour primaries. This is a synonym for palette (see definition below), but is used in this Recommendation to reference the particular cached palette associated with cached bitmap data.

3.5 desktop: The logical or physical display area for a particular terminal or window manager advertised by the application sharing conference entity (ASCE) in the application sharing (AS) capabilities. An ASCE may choose to advertise a size corresponding to the actual terminal display area or some other logical display area.

3.6 handle: An application sharing (AS) session-wide unique number used to identify an addressable item.

3.7 non-standard capability: The capability is outside the scope of this Recommendation but it has been determined through negotiation that it is recognized among all session participants.

3.8 palette: A finite set of colours defined by at least three linearly-independent colour primaries.

3.9 palettized: A term used to describe protocol elements (such as bitmap data) comprised of palettized pixels. The colour of a palettized pixel is specified by the colour value at the location in a palette referenced by the pixel value.

3.10 pointer: A bitmap that is moveable over the virtual desktop that is used as an indicator of position.

3.11 standard capability: The capability is defined within the scope of this Recommendation but is not required for all application sharing conference entity (ASCE) implementations. Note that all standard capabilities must be negotiated before use.

3.12 unicode: Multilingual text string format as defined in [ISO/IEC 10646].

3.13 virtual desktop: A logical desktop that is the largest size of all desktops of hosting application sharing conference entities (ASCEs).

3.14 window: A rectangular area on the desktop corresponding to a user interface display area managed by the terminal window manager.

3.15 window manager: A program executing on the terminal, which is responsible for managing a collection of user interface windows on the terminal desktop.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

APE Application Protocol Entity

AS Application Sharing

ASCE Application Sharing Conference Entity

ASPDU Application Sharing Protocol Data Unit

GCC Generic Conference Control

ID Identifier

MCS Multipoint Communication Service

SAP Service Access Point

5 Overview

The AS protocol enables multipoint computer application sharing by allowing a view onto a computer application executing at one site to be advertised within a session to other sites. Each site can, under specified conditions, take control of the shared computer application by sending remote keyboard and pointing device information. This style of application sharing does not require and does not make provision for synchronizing multiple instances of the same computer application running at multiple sites. Instead, it enables remote viewing and control of a single application instance to provide the illusion that the application is running locally.

An AS session consists of one or more ASCEs which cooperate via the AS protocol to share one or more applications within the session. The AS protocol defines interactions between ASCEs. It does not define interactions between an ASCE and the operating system or input and output devices on the local terminal.

5.1 Legacy and base modes

The AS protocol supports two modes of operation. Legacy mode is provided for interoperability with an existing installed base of customer terminal equipment. Base mode provides additional features that enable future AS protocol enhancement.

All ASCEs compliant with this Recommendation shall implement both legacy and base modes. The AS protocol usage of GCC (see clause 7) is defined such that where all ASCEs in a conference support base mode (i.e., there are no pre-existing legacy mode-only terminals in the conference), then all ASCEs shall use base mode. It is ITU's intention that all future enhancements to the AS protocol shall be by enhancements to base mode.

A significant proportion of this Recommendation is common to both modes, with the primary differences between the two modes being in the following areas:

- The manner in which capabilities are exchanged and negotiated: see clause 8.2.
- The manner in which ASCEs are activated and deactivated: see clause 8.4.
- The definition and encoding of ASPDUs: see clause 9.

Where there are differences between the legacy and base modes of the AS protocol, they are highlighted in the text.

5.2 AS concepts

5.2.1 Desktop and window model

This Recommendation does not assume or require any particular local terminal equipment or terminal environment, nor does it assume or require a particular local terminal window manager or windowing model. For example, it does not assume:

- that the terminal equipment is a PC or workstation – it may be a dedicated terminal;
- that AS protocol windows correspond to windows managed by a terminal window manager – they may map onto arbitrary areas of the local terminal display;
- that AS protocol desktops correspond to local terminal displays – they may map into local browser or viewer windows, or onto dedicated terminal display areas.

While this Recommendation does not assume or require any particular local terminal equipment or terminal environment, the AS protocol does define a logical desktop and window model – consisting of a collection of windows on a desktop. This requires that each active ASCE within an AS session is capable of mapping local terminal environment concepts to and from the AS protocol desktop and window model. The AS protocol desktop and window model supports the following key concepts:

- Desktop — The AS desktop is a rectangle defined in virtual desktop coordinates. An ASCE should provide a mapping from the AS desktop to an appropriate local terminal concept.
- Virtual desktop — The virtual desktop is the union of the sizes of the desktops of hosting ASCEs (i.e., ASCEs that are hosting windows – see below). The virtual desktop origin (i.e., pixel 0,0) is defined as being at (top, left).
- Window — An AS window is a rectangle defined in virtual desktop coordinates. AS windows may be wholly within, partially within or wholly outside the virtual desktop. An ASCE should provide a mapping from AS windows to an appropriate local terminal window concept.
- Z-order — The AS window Z-order defines a window depth ordering between windows on the virtual desktop, such that for two windows, the window higher in the Z-order is in front of and/or may obscure the other.
- Top-most — The top-most AS window in the Z-order is in front of and/or may obscure all other windows in the Z-order.
- Bottom-most — The bottom-most AS window in the Z-order is behind and/or may be obscured by all other windows in the Z-order.

Further AS protocol concepts are explained below or in the appropriate protocol description clauses.

Figure 2 shows an example collection of ASCEs within an AS session – this figure is used throughout this clause to illustrate key AS protocol concepts.

- ASCE A is full function, viewing and hosting – that is, it both shares windows into the AS session and displays shadow windows shared from other ASCEs. It is executing on a general purpose PC, where the AS desktop and AS windows are mapped directly onto the terminal window manager's desktop.
- ASCE B is viewing only – that is, it displays shadow windows shared from other ASCEs but does not itself share windows into the AS session. It is executing on a specialized viewing terminal where the AS desktop and AS windows are mapped into an independently sizeable browser/viewer window, which is (currently) smaller than the AS desktop and is scrolled as required to view the entire AS desktop.

- ASCE C is hosting only – that is, it shares windows into the AS session, but does not display shadow windows. It is executing on a high-end unattended workstation, where the AS desktop and AS windows are mapped directly onto the terminal window manager's desktop.

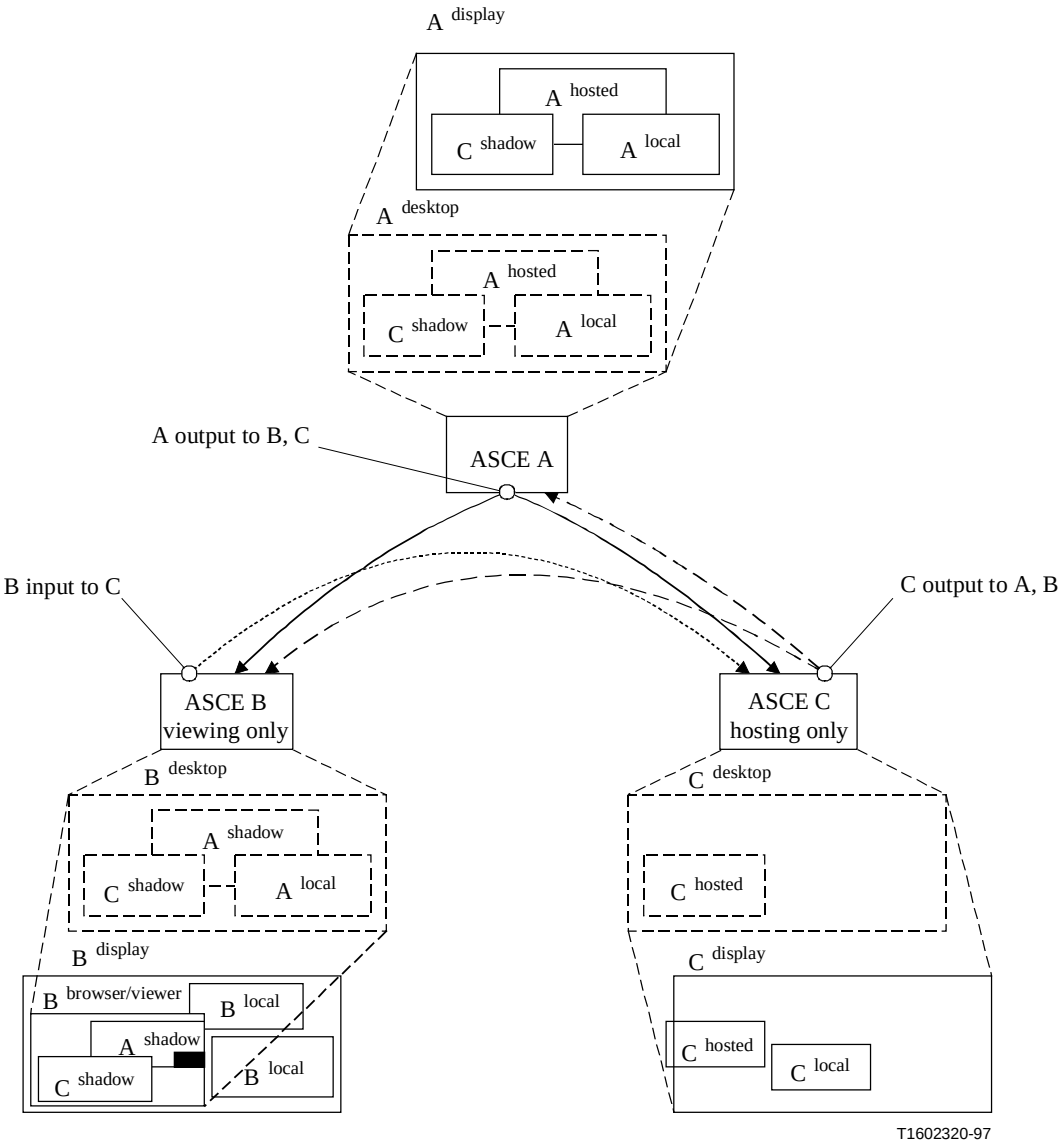


Figure 2 – Example collection of ASCEs within an AS session

Within an AS session, windows are of the following types:

- Hosted: Hosted windows are owned by an application executing on the local terminal and are shared into the AS session. For each hosted window, there will be a corresponding shadow window on each peer ASCE (except for the case where an ASCE is hosting only).
- Shadow: Shadow windows are drawn by the ASCE and correspond to a hosted window on a particular peer ASCE.
- Local: Local windows are not shared – their output is only visible on the local terminal. An ASCE is only required to track local windows where they obscure a hosted window and where that obscuring prevents the ASCE obtaining valid drawing information for the hosted window.

Figure 2 shows (for each ASCE) both the local windows on the actual terminal display and the AS windows on the AS desktop that the ASCE manages via the AS protocol.

- ASCE A is managing three windows:
  - a shadow window corresponding to the hosted window on ASCE C;
  - a hosted window which is shared into the AS session;
  - a local window – because it obscures the hosted window and on the local terminal that prevents it obtaining valid drawing information for that hosted window.

  Shadow window C is drawn on the local terminal display by the ASCE – and the other two windows are displayed by the local terminal.
- ASCE B is also managing three windows:
  - a shadow window corresponding to the hosted window on ASCE A;
  - a shadow window corresponding to the hosted window on ASCE C;
  - the local window from ASCE A – because it obscures the hosted window on ASCE A;
  - ASCE B does not track the local windows on its terminal display as it does not host windows.

  Shadow windows A and C are drawn into the browser/viewer window on the local terminal display by the ASCE. The local window is not drawn, but is rather used to compute and draw an obscuring area, which partially obscures shadow window A (shown as a dark area at the bottom right of shadow window A).
- ASCE C is managing one window:
  - a hosted window which is shared into the AS session;
  - ASCE C does not track the shadow window corresponding to the hosted window on ASCE A as it does not view shadow windows;
  - ASCE C does not track its local window, as, while the local window does obscure the hosted window, the local terminal allows it to obtain valid drawing information for that hosted window.

  The hosted window is displayed by the local terminal.

See clause 8.10 for further information on windows and window list management.

5.2.2 Output

When an ASCE is hosting a window, it is responsible for constructing an output stream for that window which will allow other ASCEs to faithfully draw corresponding shadow windows.

The AS output protocol stream is multicast via the AS-CHANNEL to all other active ASCEs within the conference. If an ASCE is hosting only (as is ASCE C in Figure 2), then it can ignore some or all of the output stream. Similarly, a receiving ASCE may decide to draw only a subset of the output stream based on its source (e.g., only draw windows from a particular ASCE).

An ASCE determines the allowable output repertoire based on the current negotiated capabilities. This requires that the ASCE is capable of adjusting its output strategy as ASCEs with lower or higher capabilities join and/or leave the conference. But within the current allowable repertoire, and the inherent constraints of the AS protocol, an ASCE has considerable freedom in constructing an output stream.

The output stream consists of:

- state information – such as window list information (see clause 8.10);
- color information – such as palette and cached ColorTable information (see clauses 8.15 and 8.16.8);

- orders and bitmap data (see clauses 8.16 and 8.17).

The AS protocol supports the following primary orders:

- Destination Blt.
- Pattern Blt.
- Screen Blt.
- Memory Blt.
- Memory three-way Blt.
- Text.
- Extended text.
- Frame.
- Rectangle.
- Line.
- Opaque rectangle.
- Desktop save.
- Desktop origin.

In typical application sharing scenarios, orders constitute a very high proportion of ASPDU traffic. Orders may therefore be differentially encoded to reduce the volume of orders in the output stream – this process is referred to as order encoding.

The AS protocol supports a single bitmap data format, which may be compressed using an AS-specific two-dimensional run-encoding bitmap compression algorithm.

Depending on local application drawing activity into hosted windows and on flow control (see clause 8.5), a hosting ASCE may use a range of strategies to reduce the volume of the output stream and/or retain responsiveness of remote drawing. Possible strategies include the following:

- Switching between sending orders and bitmap data:

  Where a local application is very actively drawing into a hosted window and the ASCE is experiencing flow control back pressure, the ASCE may prefer to accumulate bounds information for the drawing activity, rather than sending the orders, and then subsequently send bitmap data for the accumulated bounds. This may reduce the frequency of updates at remote ASCEs, but will minimize the data in flight and ensures that remote ASCEs "keep up" with the drawing activity. The switching points between orders and bitmap data and vice versa will depend on application drawing behaviour, flow control and the particular ASCE implementation.

- Collapsing overlapping updates – this process is referred to as spoiling:

  Where the ASCE is experiencing flow control back pressure and the local application draws a sequence of overlapping or occluding order or bitmap data updates, then the sequence may (in some cases) be collapsed into a smaller sequence and/or a single update. Again, this may reduce the frequency of updates at remote ASCEs, but will minimize the data in flight and ensure that remote ASCEs "keep up" with the drawing activity.

A hosting ASCE is required to ensure that the final constructed output stream allows receiving ASCEs to correctly draw the corresponding shadow windows. Similarly, where the output stream relies on control information (such as window list or palette information) that has changed on the hosting ASCE, it is the hosting ASCE's responsibility to ensure that the control information is sent prior to the output. All order-dependent control and output ASPDUs are sent at low priority, so that the sending ASCE can reliably order such output.

5.2.3 Control and input

Within an AS session, control is managed via a combination of conductorship, the core control protocol and (where negotiated) the mediated control protocol – see clauses 8.19, 8.12 and 8.13, respectively. Conductorship has preference, in that where the conference is in conducted mode, the AS-specific control protocols are not used (other than to enforce conductorship). Where conductorship is not in force, then control is managed via a combination of the core control protocol and (where negotiated) the mediated control protocol.

A key common concept to these control schemes is that, at any point in time within the AS session, one ASCE is in control and has the right to provide input to hosted and/or shadow windows[1].

Where an ASCE is in control and can provide input to hosted or shadow windows (depending on whether it or other ASCEs are detached or not), then, when a local input event occurs, the ASCE determines, based on its local window structure and the shared window structure on its logical desktop, whether that event is for a local, hosted or shadow window. Where the event is for a local or hosted window, then the ASCE can defer processing of the event to the local terminal. Where the event is for shadow window, then the ASCE determines which peer ASCE owns the corresponding hosted window and constructs a suitable input stream for that ASCE.

The AS input stream consists of interleaved keyboard and pointing device events. Keyboard events may be represented either as codepoints or as virtual keycodes. Pointing device events are represented as a logical three-button pointing device. See clause 8.18 for further information.

Where an ASCE is experiencing flow control back pressure (see clause 8.5) and there is a high level of end-user pointing device activity, the ASCE may collapse a sequence of pointing device move events into a single move – this process is referred to as input spoiling. Input spoiling may also be applied at the hosting ASCE, where a sequence of pointing device events is queued within the ASCE awaiting injection into the local terminal environment. Both send and receive input spoiling may reduce the frequency of pointing device move updates presented to hosted windows and therefore degrade the smoothness of pointing device movement, but will minimize the data in flight and ensure that remote ASCEs "keep up" with the pointing device activity.

5.2.4 Colour

The AS protocol carries colour information for the processing of orders and bitmap data.

- For orders (with the exception of the memory Blt and memory three-way Blt orders – see below) colour information is embedded in the order itself – although it may be absent on individual orders as a result of order encoding.
- Bitmap data is always palettized and is interpreted with reference to a previously sent palette which contains the colour information.
- For the memory Blt and memory three-way Blt orders, the order contains some of the required colour information, but also refers to cached bitmap data (as the source for the operation), where that cached bitmap data (see clause 8.16.7) is always palettized and is interpreted with reference to a previously cached ColorTable (see clause 8.16.8) which contains the colour information.

In the legacy mode of the AS protocol, colour information is expressed solely as RGBs. In base mode, colour information is expressed as RGBs with optional colour accuracy information.

---

[1] This Recommendation does not specify control behaviour with respect to local ASCE windows.

ASCEs may supply colour space information:

- within palettes: where the colour space only applies to that palette;
- within ColorTables in cache ColorTable orders: where the colour space only applies to that cached ColorTable;
- in colour space orders: where the colour space applies to subsequent orders.

The AS protocol does not mandate a particular colour mapping between the ASCE and the local terminal, although ASCEs should attempt to maintain colour fidelity (within the constraints of the negotiated capabilities) when constructing and interpreting colour-related information in the output stream.

5.2.5 Coordinates and clipping

All coordinates in the AS protocol are in virtual desktop coordinates. The virtual desktop is the union of the sizes of the desktops of hosting ASCEs (i.e., ASCEs that are hosting windows) and is regarded as the common drawing area in use by hosting ASCEs. The virtual desktop origin (i.e., pixel 0,0) is defined as being at (top, left).

An ASCE shall clip all AS protocol coordinates and rectangles to the virtual desktop. The AS protocol does not mandate a particular clipping strategy with respect to the local terminal display area.

All rectangles in the AS protocol are inclusive. That is, the right-most and bottom-most pixel coordinates represent pixels that are within the rectangle. For example, a rectangle with coordinates (10,10,19,19) is 10 by 10 pixels in size.

6 Use of MCS

All T.128 communication shall be through MCS as specified in [ITU-T T.122]. This clause details specific use of MCS services, channel allocation and data priorities. This Recommendation complies with the mechanisms described in [ITU-T T.121] regarding proper operations for standard base sessions, non-standard base sessions and the registration session. All other session types are optional.

An ASCE uses the MCS service primitives described in Table 6-1 to attach and detach from a domain, join and leave the AS channel, and send and receive ASPDUs.

Table 6-1 – MCS primitives needed by an ASCE

MCS primitive	Description
MCS-ATTACH-USER	Creates an MCS attachment through an MCS SAP to a domain hosted by the MCS provider. A result is confirmed to the requester. If the request is accepted, a user ID is assigned.
MCS-DETACH-USER	Deletes an MCS attachment that was created previously by invocation of MCS-ATTACH-USER. This primitive may be requested by a user or initiated by a provider. It delivers an indication at every other MCS attachment to the same domain. If provider-initiated, an indication is also delivered at the deleted attachment.
MCS-CHANNEL-JOIN	Used by an application client to join an appropriate channel whose use is defined by the application. This is a prerequisite for receiving data sent to the channel.

Table 6-1 – MCS primitives needed by an ASCE

MCS primitive	Description
MCS-CHANNEL-LEAVE	Used by an application client to leave a previously joined channel and thus stop receiving data sent to that channel. The primitive may be user-initiated (request only) or provider-initiated (indication to affected user only).
MCS-SEND-DATA	Used to transmit data to other members of a domain. If the sender is a member of the destination channel, it will not receive its own data indications. However, it will receive data indications from other sources addressed to that channel.

MCS request primitives are directed from the ASCE to the MCS provider, while indication primitives are directed from the MCS provider towards the ASCE. Additional detail on the MCS primitives described above can be found in [ITU-T T.122].

6.1 MCS channel usage

Table 6-2 describes MCS channel usage for ASCE sessions of the types defined in [ITU-T T.121]. In the case of a standard base session (see [ITU-T T.121]) which uses the base mode of the AS protocol, the channel IDs shown in Table 6-2 shall be used (symbolic IDs shown). For all other session types, the application registry resource IDs shown in the table shall be used for allocating dynamic channels. The given resource IDs shall be encoded as three-octet T.50 text strings using the characters shown in quotes in Table 6-2.

Table 6-2 – Description of AS channels

Mnemonic	Channel IDs for static channel	Application registry resource IDs for dynamic channels	Description
AS-{MCS-USER-ID}-CHANNEL	–	–	Certain ASPDUs are sent directly to individual ASCEs. To do this, the individual MCS-USER-ID channels of the peer ASCEs in the MCS domain are used.
AS-CHANNEL	AS-CHANNEL-0	"421"	This channel bears all ASPDUs to be broadcast to all peer ASCEs in a domain.

6.2 Use of MCS data services

Table 6-3 lists the use of the MCS data service MCS-SEND-DATA for each ASPDU (with mode-specific ASPDUs indicated where necessary). ASCEs do not use the MCS data service MCS-UNIFORM-SEND-DATA. This table includes the channel over which the data are sent and the data priority at which the data are sent.

The AS protocol uses three MCS priorities as a means of segregating ASPDUs with respect to flow control. Flow control is applied on a per-channel and per-priority basis (see clause 8.5), such that grouping ASPDUs by priority allows the application of a flow control regime that is applicable to the group, as follows:

- High priority is flow controlled and is used for activation, flow control and input ASPDUs.
- Medium priority is not flow controlled and is used for control ASPDUs.

- Low priority is flow controlled and is used for window information and output ASPDUs.

All ASPDUs specified in this Recommendation are placed in the data parameter of the MCS-SEND-DATA primitive. The ASPDUs are packed into the sequence of octets that form the data parameter such that the leading bit is placed in the most significant bit of each octet, and filled toward the least significant bit of the octet.

Table 6-3 – Use of MCS data primitives for ASPDUs

ASPDU	Channel	Modes	Priority
ApplicationPDU	AS-CHANNEL or AS-{MCS-USER-ID}-CHANNEL (Note 1)	Both	Medium
ConfirmActivePDU	AS-CHANNEL	Legacy mode	All (Note 2)
ControlPDU	AS-CHANNEL	Both	Medium
DeactivateAllPDU	AS-CHANNEL	Legacy mode	High
DeactivateOtherPDU	AS-{MCS-USER-ID}-CHANNEL	Legacy mode	High
DeactivateSelfPDU	AS-CHANNEL	Legacy mode	High
DemandActivePDU	AS-CHANNEL	Legacy mode	High
FlowResponsePDU	AS-{MCS-USER-ID}-CHANNEL	Both	High
FlowStopPDU	AS-CHANNEL	Both	High
FlowTestPDU	AS-CHANNEL	Both	High, medium or low (Note 3)
FontPDU	AS-CHANNEL	Both	Medium
InputPDU	AS-CHANNEL	Both	High
MediatedControlPDU	AS-CHANNEL or AS-{MCS-USER-ID}-CHANNEL (Note 4)	Both	Medium
PointerPDU	AS-CHANNEL	Both	Medium
RemoteSharePDU	AS-{MCS-USER-ID}-CHANNEL	Both	Medium
RequestActivePDU	AS-{MCS-USER-ID}-CHANNEL	Legacy mode	High
SynchronizePDU	AS-CHANNEL	Both	All (Note 5)
UpdateCapabilityPDU	AS-{MCS-USER-ID}-CHANNEL	Legacy mode	Medium
UpdatePDU	AS-CHANNEL	Both	Low

Table 6-3 – Use of MCS data primitives for ASPDUs

ASPDU	Channel	Modes	Priority
WindowActivationPDU	AS-CHANNEL or AS-{MCS-USER-ID}-CHANNEL (Note 6)	Both	Low
WindowListPDU	AS-CHANNEL	Both	Low
NOTE 1 – The ApplicationPDU (NotifyHostedApplications) is sent on the AS-CHANNEL and the ApplicationPDU (UnhostApplication) is sent on a particular AS-{MCS-USER-ID}-CHANNEL. See clause 8.9 for further information. NOTE 2 – The ConfirmActivePDU is sent on all priorities. See clause 8.4.1 for further information. NOTE 3 – The FlowTestPDU is sent on the particular priority for which flow control is being applied. See clause 8.5 for further information. NOTE 4 – The MediatedControlPDU is sent on either the AS-CHANNEL or a particular AS-{MCS-USER-ID}-CHANNEL. See clause 8.13 for further information. NOTE 5 – The SynchronizePDU is sent on one of the high, medium or low priorities, as required. See clause 8.6 for further information. NOTE 6 – WindowActivationPDU indications are sent on the AS-CHANNEL and WindowActivationPDU requests are sent on a particular AS-{MCS-USER-ID}-CHANNEL. See clause 8.11 for further information.			

7 Use of GCC

The legacy mode of the AS protocol shall use the procedures of a non-standard base session in the manner specified in [ITU-T T.121] and shall use the application protocol key defined in Annex B. The base mode of the AS protocol may use the procedures defined for a registration session, a standard base session, a public session, or a private session in the manner specified in [ITU-T T.121] and shall use as its application protocol key the object identifier defined in Annex C.

All ASCEs compliant with this Recommendation shall first enrol actively or inactively in the registration session (to signal support for the base mode of the AS protocol) and then enrol actively or inactively in the non-standard base session (to signal support for the legacy mode of the AS protocol), in both cases using the procedures defined in [ITU-T T.121], and shall stay enrolled in both sessions for as long as support for the AS protocol is to be indicated.

Pre-existing terminal equipment that only provides support for legacy mode will only enrol actively or inactively in the non-standard base session using the procedures defined in [ITU-T T.121] and will stay enrolled in that session for as long as such support is to be indicated.

An ASCE shall not enrol in the standard base session of the AS protocol unless for each node in a conference, there are an equal number of ASCEs enrolled either actively or inactively in the non-standard base session as there are enrolled in the registration session.

If the non-standard base session is in progress (i.e., all ASCEs are active in legacy mode), upon receiving a GCC-Conference-Roster-Report indication where the number of ASCEs enrolled in the registration session is equal to the number of ASCEs enrolled in the non-standard base session in the conference (i.e., all ASCEs in the conference are compliant), all ASCEs shall become inactive in legacy mode, shall enrol in the standard base session and shall become active in base mode. See clause 8.4 for further information on ASCE activation.

If the standard base session is in progress (i.e., all ASCEs are active in base mode), upon receiving a GCC-Conference-Roster-Report indication where the number of ASCEs enrolled in the registration session is less than the number of ASCEs enrolled in the non-standard base session in the conference (i.e., one or more ASCEs in the conference are non-compliant pre-existing terminal

equipment), all ASCEs enrolled in the standard base session shall become inactive in base mode, shall unenroll from the standard base session and shall become active in legacy mode. See clause 8.4 for further information on ASCE activation.

The above rules for switching between legacy and base modes mean that legacy mode will only be in effect where the conference contains pre-existing terminal equipment supporting the legacy mode of the AS protocol only. Where all ASCEs in the conference comply with this Recommendation, then base mode will be in effect. Given this, the switch from base to legacy mode occurs when the first legacy-only mode node joins a conference consisting entirely of ASCEs that are compliant with this Recommendation. In contrast, the switch from legacy mode to base mode occurs when the last legacy only mode node leaves a conference which then consists entirely of ASCEs that are compliant with this Recommendation.

ASCEs supporting the base mode of the AS protocol may enrol in a public session or private session at their discretion using the procedures defined in [ITU-T T.121].

8 Protocol specification

8.1 AS sessions

An AS session consists of one or more ASCEs enrolled within a conference as described in clause 7. ASCEs may join or leave the AS session at any time.

8.2 Capabilities

AS capabilities are collected into capability sets, where each set consists of related individual capabilities. Capability sets are described in Tables 8-3 through 8-20. Hereafter, where this Recommendation makes reference to an individual capability, it uses the notation **capability_set.capability**. For example Bitmap.desktopWidth refers to the desktopWidth capability in the Bitmap capability set.

The full list of capability sets is referred to as an ASCE's combined capabilities. The combined capabilities list contains one copy of each of the corresponding capability sets in any order.

- General capability set: See clause 8.2.3.
- Bitmap capability set: See clause 8.2.4.
- Order capability set: See clause 8.2.5.
- Bitmap cache capability set: See clause 8.2.7.
- ColorTable cache capability set: See clause 8.2.8.
- Window activation capability set: See clause 8.2.9.
- Control capability set: See clause 8.2.10.
- Pointer capability set: See clause 8.2.11.
- Share capability set[2]: See clause 8.2.12.

In the legacy mode of the AS protocol, an ASCE may define non-standard capability extensions by adding private capability sets and/or by adding private capabilities at the end of the defined capability sets. The interpretation of such capabilities is not covered by this Recommendation. When using the legacy mode of the AS protocol, an ASCE should ignore unrecognized non-standard capability sets and unrecognized capability set extensions. Similarly, an ASCE shall

[2] The share capability set is only defined for the legacy mode of the AS protocol. It does not form part of an ASCE's combined capabilities in the base mode of the AS protocol.

treat any private capabilities that are not supplied by other ASCEs as zero (for integer values) or FALSE (for logical and bit flag values).

In the base mode of the AS protocol, an ASCE may define non-standard capability extensions by adding one or more non-standard capability sets (see clause 8.2.13) to its combined capabilities. The interpretation of such capabilities is not covered by this Recommendation.

8.2.1 Distribution of capabilities

For the legacy mode of the AS protocol, an ASCE advertises its combined capabilities during ASCE activation on the following ASPDUs.

- DemandActivePDU: See clause 8.4.1.
- RequestActivePDU: See clause 8.4.1.
- ConfirmActivePDU: See clause 8.4.1.

For the legacy mode of the AS protocol, an ASCE advertises a change in a particular capability set using the UpdateCapabilityPDU (see clause 8.2.14).

For the base mode of the AS protocol, capabilities exchange shall be performed according to [ITU-T T.121], via the application enrolment mechanism. An ASCE shall use GCC-Application-Enroll requests to enrol and advertise its current combined capabilities. Where an ASCE's capabilities change while enrolled, it shall re-enrol by issuing a GCC-Application-Enroll request with the Enroll/Unenroll flag set to Enroll, including the revised combined capabilities. ASCEs receive collapsed and non-collapsed application capabilities, generated from the combined capabilities supplied by all enrolled ASCEs, via GCC-Application-Roster-Report indications. An ASCE may be required to process GCC-Application-Roster-Report indications multiple times during an AS session and shall adhere to the bounds imposed by the capabilities reported in this fashion.

8.2.2 Capabilities negotiation

In the legacy mode of the AS protocol, the ASCE activation ASPDU exchanges (see clause 8.4.1) ensure that an ASCE has a copy of each other active ASCE's capabilities. And, in addition, an ASCE is responsible for performing all capabilities negotiation. That is, GCC is not involved in capabilities distribution or negotiation.

In the base mode of the AS protocol, capabilities are distributed by GCC. There are a small number of capabilities that must be advertised via the non-collapsing capabilities list in the roster – they are specified as such in the description of the capability in the appropriate base mode capability set. However, in general, a particular capability may be advertised via one of either the collapsing or non-collapsing capabilities lists in the roster. Where an ASCE advertises a particular capability via one list, it shall not advertise the same capability via the other list. However, an ASCE is not required to advertise a capability – it may choose not to advertise a particular capability in either list and rely on the defined GCC roster counting behaviour to force use of a default value (see below).

This hybrid approach to base mode capabilities distribution is provided so that ASCEs may advertise capabilities via the collapsing capabilities, which results in a reduction in roster-related GCC traffic, but may still advertise capabilities via the non-collapsing capabilities where that is more appropriate. ASCEs are recommended to use collapsing capabilities wherever possible.

In the base mode of the AS protocol, where an ASCE advertises a particular capability via the non-collapsing capabilities list in the roster, the ASCE shall encode the capability value using the encoding rules defined in clause 9.4.

ASCE capabilities negotiation for a particular capability depends on the defined capability negotiation rule for that capability, the protocol mode (i.e., whether the ASCE is using legacy or

base mode) and whether (in base mode) the capability is advertised via collapsing or non-collapsing capabilities.

8.2.2.1 One and info capability rules

Where the defined capability negotiation rule is one or info, then an ASCE shall process the advertised capabilities as follows.

In base mode, where a capability is negotiated using either the one or info rules, the capability shall be advertised using non-collapsing capabilities.

Rule	Required ASCE processing
One	The negotiated capability value is the value advertised by a particular peer ASCE.
Info	Capabilities may also be classified as informational – they are provided solely for informational or diagnostics purposes and are not negotiated.

8.2.2.2 Min and max capability rules

Where the defined capability negotiation rule is min or max, then an ASCE shall process the advertised capabilities based on the number of candidate values, as follows:

- For the legacy mode of the AS protocol, candidate values correspond to the advertised values supplied by each active ASCE during ASCE activation, but excluding the value advertised by the negotiating ASCE. If there are N active ASCEs, then there are N–1 candidate values.
- For the base mode of the AS protocol, candidate values consist of the collapsed capability value provided in the roster and any non-collapsing values, but excluding a non-collapsing value advertised by the negotiating ASCE. The roster indicates how many ASCEs advertised capability values contributed to the collapsed value, which allows an ASCE to determine whether all ASCEs advertised a value. So, if there are N ASCEs in the roster, C ASCEs advertised collapsing values and NC ASCEs advertised non-collapsing values for a particular capability, then:
  - if N = C, all ASCEs advertised collapsing values for the particular capability: there is one candidate value;
  - if N = NC, all ASCEs advertised non-collapsing values for the particular capability: there are NC–1 candidate values – where the non-collapsing value advertised by the negotiating ASCE is excluded;
  - if N = C + NC, all ASCEs advertised either a collapsing or non-collapsing value for the particular capability: there are:
    - C + NC–1 candidate values where the negotiating ASCE advertised a non-collapsing value which is excluded;
    - C + NC candidate values where the negotiating ASCE advertised a collapsing value;
  - if N > C + NC, not all ASCEs advertised a value: the negotiated capability value is the default value (see below).

In base mode, if an ASCE determines that all ASCEs have advertised collapsing values (i.e., case N = C above), then the negotiated capability value is the collapsed value in the roster.

In base mode, if an ASCE determines that not all ASCEs advertised a value (i.e., case N > C + NC above), then the negotiated capability value is the default value. For logical capabilities, the default value is FALSE (or not established). For integer capabilities, the default value is specified in the description of the capability in the appropriate base mode capability set.

For legacy mode, or for base mode where all ASCEs advertised values and the number of candidate values is greater than one (i.e., some or all ASCEs advertised non-collapsing values – cases N = NC and N = C + NC above), then the ASCE applies the defined min or max capability negotiation rule, as follows.

Rule	Required ASCE processing
Min	The negotiated capability value is the minimum of the candidate values: • For integer values, the minimum is the lowest integer value. • For logical values, the minimum is FALSE. • For bit flags (in legacy mode only), each bit flag is negotiated independently and the minimum is not set.
Max	The negotiated capability value is the maximum of the candidate values: • For integer values, the maximum is the highest integer value. • For logical values, the maximum is TRUE. • For bit flags (in legacy mode only), each bit flag is negotiated independently and the maximum is set.

8.2.2.3 Group capability rules

ASCE capabilities negotiation is generally performed on single capability values. However, there are cases where a number of capabilities are negotiated as a group – for example, negotiation of the bitmap capability set bits-per-pixel related capabilities (see clause 8.2.4). Where this is the case, an individual capability in the group may be negotiated either by using one of the previously described capability negotiation rules – such as min or max – with the negotiated value being fed into the group negotiation, or by feeding its unnegotiated value directly into the group negotiation.

Where capabilities are negotiated as a group, they are identified as such in the particular capability set together with a reference to the specific rule used to negotiate the group.

Table 8-1 summarizes the notation used in the remainder of this clause when describing capabilities.

Table 8-1 – Capability notation – Classes

Class	Description
L	Logical: A logical value. In legacy mode, the value is either TRUE or FALSE. In base mode, if the capability is supplied, the value is TRUE. In base mode, the default value is FALSE.
F	Bit flags: A collection of bit values, each of which is TRUE or FALSE. This class is only allowed in legacy mode capability sets.
N	Integer: A signed or unsigned integer value. In base mode, the default value is as specified in the description of the capability set.
S	String: A null-terminated T.50 text string. This class is only allowed in legacy mode capability sets.

Table 8-2 illustrates example capability negotiations. Each row shows, for each ASCE (ASCEs 1 and 2), the capability value set (for each class) by the ASCE when advertising its capabilities, the resultant values for the one rule with respect to the other ASCEs and the negotiated value for the min and max rules for the cases where all ASCEs advertise using collapsing capabilities and where all advertise using non-collapsing capabilities (which gives four combinations: min (C) is the min

rule and all advertise using collapsing capabilities, min (NC) is the min rule and all advertise using non-collapsing capabilities, and so on ...)[3]. For example:

- ASCE 1's advertised logical value is TRUE, its bit flag value is 0x0001 and its integer value is 100.
- ASCE 2's advertised logical value is FALSE, its bit flag value is 0x0003 and its integer value is 300.
- When ASCE 1 applies the one rule with respect to ASCE 2, it generates ASCE 2's advertised values – i.e., FALSE, 0x0003 and 300 respectively.
- When ASCE 1 applies the max rule for the case where both ASCEs advertised using collapsing capabilities, it generates the maximum of all advertised values – i.e., TRUE and 300 respectively.
- When ASCE 1 applies the max rule for the case where both ASCEs advertised using non-collapsing capabilities, it generates the maximum of all advertised values excluding its own – i.e., FALSE, 0x0003 and 300 respectively.
- When ASCE 2 applies the one rule with respect to ASCE 1, it generates ASCE 1's advertised values – i.e., TRUE, 0x0001 and 100 respectively.

This is not an exhaustive example. The itemized cases cover four out of the ten example cases. In addition, there are further combinations where ASCEs provide mixed collapsing and non-capabilities.

Table 8-2 – Example capability negotiation

	ASCE 1						ASCE 2					
Class	**Value**	**One ⇒2**	**Min (C)**	**Min (NC)**	**Max (C)**	**Max (NC)**	**Value**	**1⇐ One**	**Min (C)**	**Min (NC)**	**Max (C)**	**Max (NC)**
B	T	F	F	F	T	F	F	T	F	T	T	T
F	x0001	x0003	n/a	x0003	n/a	x0003	x0003	x0001	n/a	x0001	n/a	x0001
N	100	300	100	300	300	300	300	100	100	100	300	300

8.2.3 General capability set

The general capability set provides capabilities for the general characteristics of the issuing ASCE. See Tables 8-3 and 8-4.

Table 8-3 – General capability set (legacy mode)

Capability	Description	Class	Rule
OSMajorType	This capability indicates the operating system major type. Allowable values are as follows: • Unspecified. • Windows. • OS/2. • Macintosh. • UNIX/X. This capability is for information and diagnostic purposes only.	N	Info

[3] The bit flag class is only allowed in legacy mode, where the distribution of capabilities via the ASCE activation ASPDUs is equivalent to advertising via non-collapsing capabilities. Therefore, the bit flag class resultant values for min and max collapsing capability cases are not shown.

Table 8-3 – General capability set (legacy mode)

Capability	Description	Class	Rule
OSMinorType	This capability indicates the operating system minor type. The values depend on the OSMajorType capability (see above). Allowable values are as follows: OSMajorType — OSMinorType Windows — Unspecified; Windows 3.1x; Windows 95; Windows NT OS/2 — Unspecified; OS/2 Warp (Intel x86); PowerPC Macintosh — Unspecified; Macintosh; PowerPC UNIX/X — Unspecified; Native server; Pseudo server Unspecified — Unspecified This capability is for information and diagnostic purposes only.	N	Info
protocolVersion	This capability specifies the protocol version level. The allowable value is 0x0200 (indicating major and minor versions of 2 and 0 respectively).	N	Info
generalCompressionTypes	This capability is a set of bit flags itemizing which (if any) non-standard general compression schemes are supported by this ASCE. Interpretation of this field depends on the negotiated generalCompressionLevel capability (see below) as follows: • If the negotiated generalCompressionLevel capability is zero, then only the least significant bit flag of this field is valid (i.e., bit 0). • If the negotiated generalCompressionLevel capability is greater than zero, then all bit flags within this field are valid. See clause 8.3.2 for further information on non-standard general compression.	F	Min
updateCapabilityFlag	This capability indicates whether this ASCE can receive the UpdateCapabilityPDU. A value of TRUE indicates that it can receive the UpdateCapabilityPDU and a value of FALSE indicates that it cannot. See clause 8.2.14 for further information.	L	One
remoteUnshareFlag	This capability indicates whether this ASCE can receive an ApplicationPDU with the UnhostApplication action. A value of TRUE indicates that it can receive an ApplicationPDU with the UnhostApplication action and a value of FALSE indicates that it cannot. See clause 8.9 for further information.	L	One

Table 8-3 – General capability set (legacy mode)

Capability	Description	Class	Rule
generalCompressionLevel	This capability indicates which level of general compression scheme handling is supported by this ASCE. See clause 8.3.2 for further information on non-standard general compression.	N	Min

Table 8-4 – General capability set (base mode)

Capability (default value)	**Description**	**ID**	**Class**	**Rule**
remoteUnshareFlag	This capability indicates whether this ASCE can receive an ApplicationPDU with the UnhostApplication action. See clause 8.9 for further information. This capability must be advertised using non-collapsing capabilities.	1	L	One
v42bisCompressionFlag	This capability indicates whether this ASCE can receive ASPDUs general compressed using V.42 *bis* compression. See clause 8.3.2 for further information.	2	L	Min
v42bisNumberCodeWords (default: 512)	This capability specifies the total number of codewords to be used by the V.42 *bis* compression algorithm. This is an upper bound on V.42 *bis* parameter P1. [ITU-T V.42 *bis*] does not impose an upper limit on its value. See clause 8.3.2 for further information.	3	N	Min (Note)
v42bisMaxStringLength (default: 6)	This capability specifies the maximum string length input to the V.42 *bis* encoder. This is an upper bound on V.42 *bis* parameter P2. See clause 8.3.2 for further information.	4	N	Min (Note)
NOTE – The v42bisNumberCodeWords and v42bisMaxStringLength capabilities are dependent on the v42bisCompressionFlag capability. If the respective numberOfEntities parameter returned by the GCC-Application-Roster-Report indication for the v42bisNumberCodeWords or v42bisMaxStringLength capabilities is equal to the numberOfEntities parameter for the v42bisCompressionFlag capability, then the respective negotiated value is established; otherwise the default value of the capability is established.				

8.2.4 Bitmap capability set

The bitmap capability set provides capabilities for the bitmap-oriented characteristics of the issuing ASCE. See Tables 8-5 and 8-6.

Table 8-5 – Bitmap capability set (legacy mode)

Capability	Description	Class	Rule
preferredBitsPerPixel	This capability indicates this ASCE's preferred format for bitmap data. Allowable values are 1, 4 and 8.	N	Group (Note 1)

Table 8-5 – Bitmap capability set (legacy mode)

Capability	Description	Class	Rule
receive1BitPerPixelFlag	This capability indicates whether this ASCE can receive 1 bit-per-pixel bitmap data. An ASCE is required to be able to receive 1 bit-per-pixel bitmap and shall set this parameter to TRUE.	L	Group (Note 1)
receive4BitsPerPixelFlag	This capability indicates whether this ASCE can receive 4 bits-per-pixel bitmap data. A value of TRUE indicates that it can receive 4 bits-per-pixel bitmap data and a value of FALSE indicates that it cannot. Where an ASCE specifies that it can receive 4 bits-per-pixel, then it must also receive 1 bit-per-pixel.	L	Group (Note 1)
receive8BitsPerPixelFlag	This capability indicates whether this ASCE can receive 8 bits-per-pixel bitmap data. A value of TRUE indicates that it can receive 8 bits-per-pixel bitmap data and a value of FALSE indicates that it cannot. Where an ASCE specifies that it can receive 8 bits-per-pixel, then it must also receive 1 and 4 bits-per-pixel.	L	Group (Note 1)
desktopWidth	This capability specifies this ASCE's current desktop width in pixels.	N	Group (Note 2)
desktopHeight	This capability specifies this ASCE's current desktop height in pixels.	N	Group (Note 2)
desktopResizeFlag	This capability indicates whether this ASCE can receive UpdateCapabilityPDUs containing a bitmap capability set as a result of a peer ASCE desktop resize. A value of TRUE indicates that it can receive an UpdateCapabilityPDU containing a bitmap capability set and a value of FALSE indicates that it cannot. See clause 8.2.14 for further information.	L	One
bitmapCompressionFlag	This capability indicates whether this ASCE can receive compressed bitmap data in UpdatePDU (bitmap) and in cache bitmap orders. See clause 8.17 for further information on bitmap compression.	L	Min
NOTE 1 – The bits-per-pixel related capabilities are negotiated as a group to determine the sendingBitsPerPixel using the algorithm described in clause 8.2.4.1. NOTE 2 – The desktop width and desktop height capabilities are negotiated as a group to determine (among others) the size of the virtual desktop. See clause 8.2.4.2.			

Table 8-6 – Bitmap capability set (base mode)

Capability (default value)	**Description**	**ID**	**Class**	**Rule**
preferredBitsPerPixel (default: 8)	This capability indicates this ASCE's preferred format for bitmap data. Allowable values are 1, 4 and 8. This capability must be advertised using non-collapsing capabilities.	10	N	Group (Note 1)

Table 8-6 – Bitmap capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
receive4BitsPerPixelFlag	This capability indicates whether this ASCE can receive 4 bits-per-pixel bitmap data. Where an ASCE specifies that it can receive 4 bits-per-pixel, then it must also be able to receive 1 bit-per-pixel.	11	L	Group (Note 1)
receive8BitsPerPixelFlag	This capability indicates whether this ASCE can receive 8 bits-per-pixel bitmap data. Where an ASCE specifies that it can receive 8 bits-per-pixel, then it must also be able to receive 1 and 4 bits-per-pixel.	12	L	Group (Note 1)
desktopWidth (default: 640)	This capability specifies this ASCE's current desktop width in pixels. This capability must be advertised using non-collapsing capabilities.	13	N	Group (Note 2)
desktopHeight (default: 480)	This capability specifies this ASCE's current desktop height in pixels. This capability must be advertised using non-collapsing capabilities.	14	N	Group (Note 2)
bitmapCompressionFlag	This capability indicates whether this ASCE can receive compressed bitmap data in UpdatePDU (bitmap) and in cache bitmap orders. See clause 8.17 for further information on bitmap compression.	15	L	Min
NOTE 1 – The bits-per-pixel related capabilities are negotiated as a group to determine the sendingBitsPerPixel using the algorithm described in clause 8.2.4.1. NOTE 2 – The desktop width and desktop height capabilities are negotiated as a group to determine (among others) the size of the virtual desktop. See clause 8.2.4.2.				

8.2.4.1 Sending bits-per-pixel capabilities negotiation

The bitmap capability set preferredBitsPerPixel, receive1BitPerPixelFlag (in legacy mode), receive4BitsPerPixelFlag and receive8BitsPerPixelFlag capabilities are negotiated as a group to determine the sendingBitsPerPixel used by each ASCE when sending bitmap data, palettes and ColorTables.

Certain terminal types can acquire and render bitmap data in a range of colour depths, but may have a preferred colour depth (which normally corresponds to the actual terminal display colour depth) at which acquisition and rendering is either more efficient or can be achieved with better colour fidelity. Therefore, the bitmap capability set provides a range of capabilities, which enables an ASCE to express:

- its preferred colour depth for the receipt of bitmap data, palettes and ColorTables;
- other colour depths at which it can receive bitmap data, palettes and ColorTables.

The capabilities negotiation algorithm for this group determines a sendingBitsPerPixel for a particular ASCE using the advertised preferredBitsPerPixel, receive4BitsPerPixelFlag and receive8BitsPerPixelFlag capability values, as follows:

- set combinedBitsPerPixel to the minimum of this ASCE's preferredBitsPerPixel value and the maximum of all other ASCEs' preferredBitsPerPixel values;
- if the combinedBitsPerPixel is 1 then set sendingBitsPerPixel to 1;

- else if combinedBitsPerPixel is less than or equal to 4 and the negotiated receive4BitsPerPixelFlag value is TRUE then sendingBitsPerPixel is 4;
- else if the negotiated receive8BitsPerPixelFlag value is TRUE then sendingBitsPerPixel is 8;
- else if the negotiated receive4BitsPerPixelFlag value is TRUE then sendingBitsPerPixel is 4;
- else sendingBitsPerPixel is 1.

In the legacy mode of the AS protocol, the bitmap capability set contains a receive1BitPerPixelFlag capability, but this does not play a part in the negotiation, as the requirement to support lower colour depths (i.e., an ASCE is required to support at least 1 bit-per-pixel if it supports 4 bits-per-pixel and to support 1 and 4 bits-per-pixel if it supports 8 bits-per-pixel) means that all ASCEs must support 1 bit-per-pixel. In the base mode of the AS protocol, the bitmap capability set does not contain a receive1BitPerPixelFlag capability.

The requirement for an ASCE to support lower colour depths means that an ASCE conforming to this Recommendation shall not cause capabilities negotiation to fail to find a suitable value.

An ASCE may use an alternate private algorithm in certain circumstances. For example, certain 8 bits-per-pixel terminals do not reliably generate 4 bits-per-pixel bitmap data. Where this is the case and the negotiated receive8BitsPerPixel value is TRUE, the ASCE may set sendingBitsPerPixel to 8, even where the combinedBitsPerPixel value is 4. However, where an ASCE does use a private algorithm, it shall still generate a sendingBitsPerPixel value which is consistent with the negotiated values of the receive4BitsPerPixelFlag and receive8BitsPerPixelFlag capabilities.

8.2.4.2 Desktop size capabilities negotiation

The bitmap capability set desktop width and desktop height capabilities are independently negotiated to determine the size of the virtual desktop. See clause 5.2.5 for further information on the virtual desktop.

The capabilities negotiation algorithm essentially uses the max rule (see clause 8.2.2), but the candidate values are the capability values advertised by all active hosting ASCEs, which may or may not include the determining ASCE.

For example, in a conference with four active ASCEs – ASCEs A, B, C and D where:

- ASCE A advertises Bitmap.desktopWidth and Bitmap.desktopHeight as 800 and 600 respectively;
- ASCE B advertises Bitmap.desktopWidth and Bitmap.desktopHeight as 1024 and 768 respectively;
- ASCE C advertises Bitmap.desktopWidth and Bitmap.desktopHeight as 1600 and 1200 respectively;
- ASCE D advertises Bitmap.desktopWidth and Bitmap.desktopHeight as 640 and 480 respectively;
- then if ASCEs A and C are hosting, the negotiated virtual desktop capability values are 1600 by 1200;
- whereas if ASCEs B and D are hosting, the negotiated virtual desktop capability values are 1024 by 768.

8.2.5 Order capability set

The order capability set provides capabilities for the order characteristics of the issuing ASCE.

See Appendix I for informative values for the desktop save capabilities (i.e., desktopSaveXGranularity, desktopSaveYGranularity and desktopSaveSize). See Tables 8-7 and 8-8.

Table 8-7 – Order capability set (legacy mode)

Capability	Description	Class	Rule
terminalDescriptor	This capability is a null-terminated character T.50 text string which may be used to identify local terminal characteristics for information and diagnostics purposes.	S	Info
desktopSaveXGranularity	This capability specifies the minimum X granularity in pixels for this ASCE when receiving desktop save orders. See clause 8.16.17 for further information.	N	Max (Note 1)
desktopSaveYGranularity	This capability specifies the minimum Y granularity in pixels for this ASCE when receiving desktop save orders. See clause 8.16.17 for further information.	N	Max (Note 1)
maximumOrderLevel	This capability specifies the maximum order level supported within the orderSupport capability (see below). See clause 8.2.6 for further information on order levels.	N	Info
numberFonts	This capability is the maximum number of matchable fonts for this ASCE, details of which are subsequently supplied in the FontPDU. See clause 8.8 for further information on the FontPDU.	N	Info (Note 2)
orderFlags	This capability is a set of bit flags indicating the order support provided by this ASCE. Defined bit flag values are as follows. • Negotiate order support: It is mandatory to set this flag. • Cannot receive orders: If this flag is set, it indicates that this ASCE cannot receive orders. An ASCE shall always set the negotiate order support flag. See clause 8.16.3 for further information on order encoding.	F	Min (Note 3)
orderSupport	This capability is an array of 32 order levels indexed by order type. The allowable array indices are as follows. All other array values shall be set to zero. See below for further information on order levels. **Order** — **Index** Destination Blt support — 0 — See clause 8.16.4. Pattern Blt support — 1 — See clause 8.16.5. Screen Blt support — 2 — See clause 8.16.6. Memory Blt support — 3 — See clause 8.16.9. Memory three-way Blt support — 4 — See clause 8.16.10. Text support — 5 — See clause 8.16.11. Extended text support — 6 — See clause 8.16.12. Rectangle support — 7 — See clause 8.16.14.	N	Min

Table 8-7 – Order capability set (legacy mode)

Capability	Description			Class	Rule
	Line support	8	See clause 8.16.16.		
	Frame support	9	See clause 8.16.13.		
	Opaque rectangle support	10	See clause 8.16.15.		
	Desktop save support	11	See clause 8.16.17.		
textFlags	This capability is a set of bit flags indicating font matching and text options supported by this ASCE. Defined bit flag values are as follows: • Check font aspect – If this flag is set, it indicates that this ASCE supports the checking of font horizontal and vertical aspects during font matching. • Check font signatures – If this flag is set, it indicates that this ASCE supports the checking of font signatures during font matching. • DeltaX simulation – If this flag is set, it indicates that this ASCE allows Delta X approximations during font matching. • Baseline start – If this flag is set, it indicates that this ASCE can receive text and extended text orders where the text start position is specified with respect to the character baseline. See clause 8.8 for further information on font matching. See clause 8.16.11 and clause 8.16.12 for further information on text and extended text orders.			F	Min
desktopSaveSize	This capability specifies the total size in pixels of this ASCE's desktop cache per hosting ASCE. See clause 8.16.17 for further information.			N	Min (Note 1)

NOTE 1 – As a result of capabilities negotiation, an ASCE must be able to receive desktop save orders constructed using X and/or Y granularities larger than, or not exact multiples of, its advertised desktopSaveXGranularity and desktopSaveYGranularity capabilities. Similarly, an ASCE must be able to send desktop save orders where the negotiated desktopSaveSize is not an exact multiple of its advertised desktopSaveXGranularity and desktopSaveYGranularity capabilities.

NOTE 2 – A receiving ASCE may use the numberFonts capability to determine whether it has sufficient storage to receive this ASCE's fonts on subsequent FontPDUs. The numberFonts capability is an upper limit and subsequent FontPDUs may contain a number of font attributes less than or equal to the numberFonts capability. See clause 8.8 for further information on the FontPDU.

NOTE 3 – It is mandatory that the negotiate order support bit flag is set. The cannot receive orders bit flag allows an ASCE to advertise whether it can or cannot receive orders. Where the cannot receive orders bit flag is not set, it indicates that the ASCE can receive orders. Where the cannot receive orders bit flag is set, it indicates that the ASCE cannot receive orders, which forces peer ASCEs to disable the sending of order updates entirely and to just send bitmap updates.

Table 8-8 – Order capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
desktopSaveXGranularity (default: 1)	This capability specifies the minimum X granularity in pixels for this ASCE when receiving desktop save orders. See clause 8.16.17 for further information.	20	N	Max (Note)
desktopSaveYGranularity (default: 1)	This capability specifies the minimum Y granularity in pixels for this ASCE when receiving desktop save orders. See clause 8.16.17 for further information.	21	N	Max (Note)
desktopSaveSize (default: 160,000)	This capability specifies the total size in pixels of this ASCE's desktop cache. See clause 8.16.17 for further information.	22	N	Min (Note)
checkFontAspectFlag	This capability indicates whether this ASCE supports the checking of font horizontal and vertical aspects during font matching. See clause 8.8 for further information on font matching.	23	L	Min
checkFontSignaturesFlag	This capability indicates whether this ASCE supports the checking of font signatures during font matching. See clause 8.8 for further information on font matching.	24	L	Min
allowDeltaXFlag	This capability indicates whether this ASCE allows delta X approximations during font matching. See clause 8.8 for further information on font matching.	25	L	Min
baselineStartFlag	This capability indicates whether this ASCE can receive text and extended text orders where the text start position is specified with respect to the character baseline. See clause 8.16.11 and clause 8.16.12 for further information on text and extended text orders.	26	L	Min
receiveOrdersFlag	This capability indicates whether this ASCE can receive UpdatePDU (orders) ASPDUs – i.e., whether it can receive orders or not. See clause 8.16 for further information on orders.	27	L	Min
DesktopSaveLevel (default: 1)	This capability indicates the desktop save order level supported by this ASCE. See clause 8.16.17 for further information on the desktop save order.	30	N	Min
DestinationBltLevel (default: 1)	This capability specifies the destination Blt order level supported by this ASCE. See clause 8.2.6 for further information on order levels and 8.16.4 for further information on the destination Blt order.	31	N	Min

Table 8-8 – Order capability set (base mode)

Capability (default value)	**Description**	**ID**	**Class**	**Rule**
ExtendedTextLevel (default: 1)	This capability specifies the extended text order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.12 for further information on the extended text order.	32	N	Min
FrameLevel (default: 1)	This capability specifies the frame order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.13 for further information on the frame order.	33	N	Min
LineLevel (default: 1)	This capability specifies the line order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.16 for further information on the line order.	34	N	Min
MemoryBltLevel (default: 1)	This capability specifies the memory Blt order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.9 for further information on the memory Blt order.	35	N	Min
MemoryThreeWayBltLevel (default: 1)	This capability specifies the memory three-way Blt order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.10 for further information on the memory three-way Blt order.	36	N	Min
OpaqueRectangleLevel (default: 1)	This capability specifies the opaque rectangle order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.15 for further information on the opaque rectangle order.	37	N	Min
PatternBltLevel (default: 1)	This capability specifies the pattern Blt order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.5 for further information on the pattern Blt order.	38	N	Min
RectangleLevel (default: 1)	This capability specifies the rectangle order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.14 for further information on the rectangle order.	39	N	Min
ScreenBltLevel (default: 1)	This capability specifies the screen Blt order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.6 for further information on the screen Blt order.	40	N	Min

Table 8-8 – Order capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
TextLevel (default: 1)	This capability specifies the text order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.11 for further information on the text order.	41	N	Min
DesktopOriginLevel (default: 1)	This capability specifies the desktop origin order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.18 for further information on the desktop origin order.	42	N	Min
CacheBitmapLevel (default: 1)	This capability specifies the cache bitmap order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.7 for further information on the cache bitmap order.	43	N	Min
CacheColorTableLevel (default: 1)	This capability specifies the cache ColorTable order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.8 for further information on the cache ColorTable order.	44	N	Min
ColorSpaceLevel (default: 1)	This capability specifies the colour space order level supported by this ASCE. See clause 8.2.6 for further information on order levels and clause 8.16.19 for further information on the colour space order.	45	N	Min
NOTE – As a result of capabilities negotiation, an ASCE must be able to receive desktop save orders constructed using X and/or Y granularities larger than, or not exact multiples of, its advertised desktopSaveXGranularity and desktopSaveYGranularity capabilities. Similarly, an ASCE must be able to send desktop save orders where the negotiated desktopSaveSize is not an exact multiple of its advertised desktopSaveXGranularity and desktopSaveYGranularity capabilities.				

8.2.6 Order levels

The AS protocol allows for future enhancement of order support using order levels. Order level values are in the range 0..255, with the value 0 indicating that the indicated order is not supported.

The order level values in the Order.orderSupport capability (for the legacy mode of the AS protocol) and in the per-order level capability parameters (for the base mode of the AS protocol) indicate the maximum order level that the issuing ASCE can receive on an order-by-order basis. If an ASCE indicates that it can receive order level N for a particular order, then it shall be able to receive order levels in the range 1..N. It is envisaged that as additional application sharing requirements emerge (e.g., reflecting changing order usage in target terminals), then existing AS orders may be enhanced. Where that is the case, an ASCE may advertise support for a range of enhanced orders and, where it is in a conference with a group of active ASCEs that also support the enhanced orders, may use them when constructing UpdatePDU (orders) ASPDUs. In summary, this means that:

- where an ASCE supports an order at order level N, it shall be prepared to receive orders at order levels 1..N;

- where an ASCE supports an order at order level N and the negotiated order level is greater than or equal to N, then the ASCE may send the order at order levels 1..N;
- where an ASCE supports an order at order level N and the negotiated order level is less than N, then the ASCE may send the order at order levels 1 through the negotiated value – but may not send orders at order levels greater than the negotiated value.

For the legacy mode of the AS protocol, order level support is not defined for the desktop origin, cache bitmap and cache ColorTable orders (see clauses 8.16.18, 8.16.7 and 8.16.8, respectively). In the absence of order level support for these orders:

- if an ASCE supports the screen Blt order at order level 1 or above, it shall also support the desktop origin order;
- if an ASCE supports either of the memory Blt and memory three-way Blt orders at order level 1 or above, it shall also support the cache bitmap and cache ColorTable orders and shall ensure that it advertises valid values for the bitmap cache and ColorTable cache capability sets (see clauses 8.2.7 and 8.2.8 for further information on the bitmap cache and ColorTable cache capability sets).

8.2.7 Bitmap cache capability set

The bitmap cache capability set provides capabilities for the bitmap cache characteristics of the issuing ASCE. These capabilities are used to negotiate values used to construct cache bitmap orders in UpdatePDUs. See clause 8.16.7 for further information on cache bitmap orders.

If an ASCE supports either of the memory Blt and memory three-way Blt orders at order level 1 or above, it shall support the cache bitmap order (in base mode at order level 1 or above) and ensure that it advertises bitmap cache capability set values with:

- non-zero values for the cache1Entries, cache2Entries and cache3Entries capabilities;
- allowable values for the cache1MaximumCellSize, cache2MaximumCellSize and cache3MaximumCellSize capabilities (as specified in Tables 8-9 and 8-10);
- cache3MaximumCellSize > = cache2MaximumCellSize > = cache1MaximumCellSize.

Where an ASCE supports bitmap caching, these capabilities indicate the bitmap cache sizes per each other hosting ASCE. That is, by advertising these capabilities, the ASCE is committing to provide a set of bitmap caches of the advertised sizes for each other active ASCE in the conference that is hosting windows.

See Appendix I for informative values for bitmap cache capabilities.

Table 8-9 – Bitmap cache capability set (legacy mode)

Capability	Description	Class	Rule
cache1Entries	This capability is the number of cache entries in the first cache area.	N	Min
cache1MaximumCellSize	This capability is the maximum cell size in octets for the first cache area. The value for this capability is in the range 256..16384.	N	Min
cache2Entries	This capability is the number of cache entries in the second cache area.	N	Min
cache2MaximumCellSize	This capability is the maximum cell size in octets for the second cache area. The value for this capability is in the range 256..16384.	N	Min
cache3Entries	This capability is the number of cache entries in the third cache area.	N	Min

Table 8-9 – Bitmap cache capability set (legacy mode)

Capability	Description	Class	Rule
cache3MaximumCellSize	This capability is the maximum cell size in octets for the third cache area. The value for this capability is in the range 256..16384.	N	Min

Table 8-10 – Bitmap cache capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
cache1Entries (default: 600)	This capability is the number of cache entries in the first cache area.	80	N	Min
cache1MaximumCellSize (default: 496)	This capability is the maximum cell size in octets for the first cache area. The value for this capability is in the range 256..16384.	81	N	Min
cache2Entries (default: 300)	This capability is the number of cache entries in the second cache area.	82	N	Min
cache2MaximumCellSize (default: 2032)	This capability is the maximum cell size in octets for the second cache area. The value for this capability is in the range 256..16384.	83	N	Min
cache3Entries (default: 150)	This capability is the number of cache entries in the third cache area.	84	N	Min
cache3MaximumCellSize (default: 4080)	This capability is the maximum cell size in octets for the third cache area. The value for this capability is in the range 256..16384.	85	N	Min

8.2.8 ColorTable cache capability set

The ColorTable cache capability set provides capabilities for the ColorTable cache characteristics of the issuing ASCE. These capabilities are used to negotiate values used to construct cache ColorTable orders in UpdatePDUs. See clause 8.16.8 for further information on cache ColorTable orders and also Tables 8-11 and 8-12.

If an ASCE supports either of the memory Blt and memory three-way Blt orders at order level 1 or above, it shall support the cache ColorTable order (in base mode at order level 1 or above) and ensure that it advertises a ColorTable cache capability set containing a non-zero value for the colorTableCacheSize capability.

Where an ASCE supports ColorTable caching, these capabilities indicate the ColorTable cache size per each other hosting ASCE. That is, by advertising these capabilities, the ASCE is committing to provide a ColorTable cache of the advertised size for each other active ASCE in the conference that is hosting windows.

See Appendix I for informative values for ColorTable cache capability set values.

Table 8-11 – ColorTable cache capability set (legacy mode)

Capability	Description	Class	Rule
colorTableCacheSize	This capability specifies the number of ColorTable entries in this ASCE's receiving ColorTable cache. Where an ASCE supports ColorTable caching, the allowable values are in the range 1..255 (zero is not allowed). See clause 8.16.8 for further information on ColorTable caching.	N	Min

Table 8-12 – ColorTable cache capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
colorTableCacheSize (default: 6)	This capability specifies the number of ColorTable entries in this ASCE's receiving ColorTable cache. Where an ASCE supports ColorTable caching, the allowable values are in the range 1..255 (zero is not allowed). See clause 8.16.8 for further information on ColorTable caching.	90	N	Min

8.2.9 Window activation capability set

The window activation capability set provides capabilities for the window activation characteristics of the issuing ASCE, and in particular about its support for specific WindowActivationPDU activation messages. See clause 8.11 for further information on WindowActivationPDUs and also Tables 8-13 and 8-14.

Table 8-13 – Window activation capability set (legacy mode)

Capability	Description	Class	Rule
helpKeyFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the ActivationHelpKey action. A value of TRUE indicates that it can receive WindowActivationPDUs containing the ActivationHelpKey action and a value of FALSE indicates that it cannot. See clause 8.11 for further information.	L	One
helpIndexKeyFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the ActivationHelpIndexKey action. A value of TRUE indicates that it can receive WindowActivationPDUs containing the ActivationHelpIndexKey action and a value of FALSE indicates that it cannot. See clause 8.11 for further information.	L	One
helpExtendedKeyFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the ActivationHelpExtendedKey action. A value of TRUE indicates that it can receive WindowActivationPDUs containing the ActivationHelpExtendedKey action and a value of FALSE indicates that it cannot. See clause 8.11 for further information.	L	One

Table 8-13 – Window activation capability set (legacy mode)

Capability	Description	Class	Rule
windowManagerMenu Flag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the WindowManagerMenu action. A value of TRUE indicates that it can receive WindowActivationPDUs containing the WindowManagerMenu action and a value of FALSE indicates that it cannot. See clause 8.11 for further information.	L	One

Table 8-14 – Window activation capability set (base mode)

Capability (default value)	**Description**	**ID**	**Class**	**Rule**
helpKeyFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the ActivationHelpKey action. See clause 8.11 for further information. This capability must be advertised using non-collapsing capabilities.	100	L	One
helpIndexKeyFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the ActivationHelpIndexKey action. See clause 8.11 for further information. This capability must be advertised using non-collapsing capabilities.	101	L	One
helpExtendedKeyFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the ActivationHelpExtendedKey action. See clause 8.11 for further information. This capability must be advertised using non-collapsing capabilities.	102	L	One
windowManagerMenuFlag	This capability indicates whether this ASCE can receive WindowActivationPDUs containing the WindowManagerMenu action. See clause 8.11 for further information. This capability must be advertised using non-collapsing capabilities.	103	L	One

8.2.10 Control capability set

The control capability set provides capabilities for the control characteristics of the issuing ASCE. These capabilities are used to negotiate values used in the management of control and detached status between ASCEs. See clause 8.12 for further information on control characteristics and also Tables 8-15 and 8-16.

Table 8-15 – Control capability set (legacy mode)

Capability	Description	Class	Rule
controlFlags	This capability is a set of bit flags indicating the control options supported by this ASCE. Defined bit flag values are as follows: • Allow mediated control If an ASCE does not set the allow mediated control bit flag capability, the ASCE does not support the AS mediated control protocol. Where this is the case, the remoteDetachFlag, controlInterest and detachInterest capabilities need not be negotiated and the MediatedControlPDU is not valid. See clause 8.13 for further information.	F	Min
remoteDetachFlag	This capability indicates whether this ASCE allows other ASCEs to force it into detached control mode. A value of TRUE indicates that this ASCE allows other ASCEs to force it into detached control mode and a value of FALSE indicates that it does not. See clause 8.13 for further information.	L	One
controlInterest	This capability indicates this ASCE's behaviour for changes of control. Allowable values are as follows: • Always This ASCE always permits changes of control. • Confirm This ASCE requires to confirm changes of control. • Never This ASCE never allows changes of control. See clause 8.13 for further information.	N	Max (Note)
detachInterest	This capability indicates this ASCE's behaviour for changes in detach status. Allowable values are as follows: • Always This ASCE always permits changes in detach status. • Confirm This ASCE requires to confirm changes in detach status. • Never This ASCE never allows changes in detach status. See clause 8.13 for further information.	N	Max (Note)
NOTE – The controlInterest and detachInterest capabilities are negotiated using values such that never is greater than confirm which is greater than always.			

Table 8-16 – Control capability set (base mode)

Capability (default value)	**Description**	**ID**	**Class**	**Rule**
mediatedControlFlag	This capability indicates whether the issuing ASCE supports the mediated control protocol. Where this capability is not supported, the remoteDetachFlag, controlInterest and detachInterest capabilities need not be negotiated and the MediatedControlPDU is not valid. See clause 8.13 for further information.	110	L	Min

Table 8-16 – Control capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
remoteDetachFlag	This capability indicates whether this ASCE allows other ASCEs to force it into detached control mode. See clause 8.13 for further information. This capability must be advertised using non-collapsing capabilities.	111	L	One
controlInterest (default: always)	This capability indicates this ASCE's behaviour for changes of control. Allowable values are as follows. • Always This ASCE always permits changes of control. • Confirm This ASCE requires to confirm changes of control. • Never This ASCE never allows changes of control. See clause 8.13 for further information. This capability must be advertised using non-collapsing capabilities.	112	N	Max (Note)
detachInterest (default: always)	This capability indicates this ASCE's behaviour for changes in detach status. Allowable values are as follows: • Always This ASCE always permits changes in detach status. • Confirm This ASCE requires to confirm changes in detach status. • Never This ASCE never allows changes in detach status. See clause 8.13 for further information. This capability must be advertised using non-collapsing capabilities.	113	N	Max (Note)
NOTE – The controlInterest and detachInterest capabilities are negotiated using values such that never is greater than confirm which is greater than always.				

8.2.11 Pointer capability set

The pointer capability set provides capabilities for the pointer characteristics of the issuing ASCE. These capabilities are used to negotiate values used to construct PointerPDUs. See clause 8.14 for further information on pointers and also Tables 8-17 and 8-18.

Where an ASCE does not support colour pointers, or where it supports colour pointers but does not wish to support colour pointer caching, then it should advertise its Pointer.colorPointercacheSize as one. This shall be interpreted as indicating that the advertising ASCE can only remember the last monochrome and/or colour pointer (depending on the negotiated value of the Pointer.colorPointerFlag) – that is, it does not support pointer caching.

Where an ASCE supports colour pointers and colour pointer caching, these capabilities indicate the colour pointer cache size per each other hosting ASCE. That is, by advertising these capabilities, the ASCE is committing to provide a colour pointer cache of the advertised size for each other active ASCE in the conference that is hosting windows.

See Appendix I for informative values for pointer capabilities.

Table 8-17 – Pointer capability set (legacy mode)

Capability	Description	Class	Rule
colorPointerFlag	This capability indicates whether this ASCE supports colour pointers. A value of TRUE indicates that this ASCE supports colour pointers and a value of FALSE indicates that it does not. See clause 8.14 for further information.	L	Min
colorPointerCacheSize	This capability specifies the number of entries in this ASCE's receiving colour pointer cache. The allowable values are in the range 1..500. Where an ASCE does not support colour pointers, it shall set this capability to 1 (a value of zero is not allowed). See clause 8.14 for further information on pointer caching.	N	Min

Table 8-18 – Pointer capability set (base mode)

Capability (default value)	Description	ID	Class	Rule
colorPointerFlag	This capability indicates whether this ASCE supports colour pointers. See clause 8.14 for further information.	120	L	Min
colorPointerCacheSize (default: 25)	This capability specifies the number of entries in this ASCE's receiving colour pointer cache. The allowable values are in the range 1..500. Where an ASCE does not support colour pointers, it shall set this capability to 1 (a value of zero is not allowed). See clause 8.14 for further information on pointer caching.	121	N	Min

8.2.12 Share capability set

The share capability set provides information about the node on which the ASCE executes. This capability set is only supported in the legacy mode of the AS protocol.

The GCC node ID may be used by receiving ASCEs to correlate issuing ASCEs with their application records in the GCC application roster. See Table 8-19.

Table 8-19 – Share capability set (legacy mode)

Capability	Description	Class	Rule
nodeID	This capability is the GCC node ID of the advertising ASCE's node.	N	Info

8.2.13 Non-standard capability set

The non-standard capability set is used to negotiate non-standard functions. Any number of non-standard capabilities may appear in the non-standard capability set as long as they each have a unique Non-Standard-Identifier. Interpretation of these capabilities is not covered by this Recommendation.

The non-standard capability set is only supported in the base mode of the AS protocol. There is no mechanism in the legacy mode of the AS protocol for the negotiation of non-standard capabilities. See Table 8-20.

Table 8-20 – Non-standard capability set (base mode)

Capability	Description	ID	Class	Rule
nonStandardCapability	This capability is used to negotiate non-standard functions. Any number of these may appear in the ASCE base mode capabilities as long as they each have a unique Non-Standard-Identifier. The interpretation of these capabilities is not covered by this Recommendation.	Non-standard identifier	–	–

8.2.14 Capability update

For the base mode of the AS protocol, where an ASCE's capabilities change while enrolled, it shall re-enrol by issuing a GCC-Application-Enroll request to enrol in the standard base session with the enroll/unenroll flag set to enroll, including the revised combined capabilities. See clause 7 for further information on session enrolment and clause 8.2.1 for further information on the distribution of capabilities.

For the legacy mode of the AS protocol, an ASCE may advertise changes in a capability set by sending an UpdateCapabilityPDU to other ASCEs within the conference in the manner indicated in Table 6-3. The content of the UpdateCapabilityPDU is shown in Table 8-21.

An ASCE shall only send the UpdateCapabilityPDU where both the negotiated General.updateCapabilityFlag and Bitmap.desktopResizeFlag capabilities are TRUE.

The only allowable capability set that can be advertised in an UpdateCapabilityPDU is the legacy mode bitmap capability set (see clause 8.2.4).

Table 8-21 – UpdateCapabilityPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
updateCapabilitySet	This parameter shall be a bitmap capability set. See clause 8.2.4 for a description of the bitmap capability set.

When an ASCE determines that one or more capabilities have been updated, it shall perform any required capabilities negotiation (see clause 8.2.2) and shall perform hosting synchronization (if it is hosting) and shadow synchronization (see clause 8.6).

8.3 ASPDU formats

In the legacy mode of the AS protocol, all ASPDUs, other than the following flow control ASPDUs, contain a ShareControl header. Table 8-22 describes the ShareControl header for the legacy mode of the AS protocol. In the base mode of the AS protocol, ASPDUs do not contain ShareControl headers.

- FlowTestPDU: See clause 8.5.
- FlowResponsePDU: See clause 8.5.
- FlowStopPDU: See clause 8.5.

Table 8-22 – ShareControl header (legacy mode)

Parameter	Description
totalLength	This is the total length in octets of the ASPDU within which this header is included. This parameter is required as MCS implementations may segment ASPDUs in transmission and are not required to reassemble on delivery. This parameter allows receiving ASCEs to efficiently perform reassembly where MCS segmentation is present.
protocolVersion	This parameter identifies the protocol version supported by the issuing ASCE. The allowable value is 1.
PDUSource	This parameter is the MCS user ID of the ASCE sending the ASPDU containing this ShareControl header.

All ASPDUs, other than the following ASCE activation and flow control ASPDUs, contain a ShareData header and are referred to as data ASPDUs.

- DemandActivePDU: See clause 8.4.1.
- ConfirmActivePDU: See clause 8.4.1.
- RequestActivePDU: See clause 8.4.1.
- DeactivateOtherPDU: See clause 8.4.1.
- DeactivateSelfPDU: See clause 8.4.1.
- DeactivateAllPDU: See clause 8.4.1.
- FlowTestPDU: See clause 8.5.
- FlowResponsePDU: See clause 8.5.
- FlowStopPDU: See clause 8.5.

Tables 8-23 and 8-24 describe the ShareData header for the legacy and base modes of the AS protocol.

Table 8-23 – ShareData header (legacy mode)

Parameter	Description
ShareControl header	The ShareControl header is described in clause 8.3.
shareID	This parameter uniquely identifies the ASCE session within which this ASPDU is issued. See clause 8.4.2 for a description of share identifiers.
streamID	This parameter identifies the stream for this ASPDU. See clause 8.3.1 for a description of streams.
uncompressedLength	This parameter is the length of the uncompressed ASPDU data in octets, starting from and including the generalCompressedType parameter. This parameter may be used as a check on decompression. See clause 8.3.2 for further information on non-standard general compression.

Table 8-23 – ShareData header (legacy mode)

Parameter	Description
generalCompressedType	This parameter indicates whether the issuing ASCE has general compressed the ASPDU containing this ShareData header – and if so, indicates which general compression scheme type has been applied. Interpretation of this field depends on the negotiated General.generalCompressionLevel capability as follows: • If the negotiated General.generalCompressionLevel capability is zero, then the ASCE shall only reference the most significant bit of this field. • If the negotiated General.generalCompressionLevel capability is greater than zero, then the ASCE shall reference all bits within this field. A value of zero indicates that no general compression has been applied. A non-zero value specifies the particular compression type that has been applied. See clause 8.3.2 for further information on general compression.
generalCompressedLength	Where the generalCompressedType is non-zero, this parameter is the length of the compressed ASPDU data in octets, starting from after the uncompressedLength parameter. Where the generalCompressedType is zero, indicating that no general compression has been applied, this parameter shall be zero. See clause 8.3.2 for further information on general compression.

Table 8-24 – ShareData header (base mode)

Parameter	Description
shareID	This parameter is the roster instance of the ASCE session within which this ASPDU is issued. See clause 8.4.2 for a description of share identifiers.
generalCompressionSpecifier	This optional parameter identifies the general compression scheme and any associated parameters applicable to this ASPDU. See clause 8.3.2 for further information.

8.3.1 Streams

In the legacy mode of the AS protocol, an ASCE is required to send data ASPDUs over a set of logical streams. Streams are used to represent ASCE data priorities and map onto MCS priorities as described in Table 8-25. Stream identifiers are present in all legacy mode data ASPDUs. Streams are not supported in the base mode of the AS protocol.

Table 8-25 – Stream identifiers (legacy mode)

Stream ID	MCS priority
4	High
2	Medium
1	Low

8.3.2 General compression

This Recommendation specifies compression schemes that may be applied to orders (which is termed order encoding – see clause 8.16.3) and bitmap data (which is termed bitmap compression – see clause 8.17) in UpdatePDUs. Both these compression schemes deliver good compression ratios on typical AS output data for relatively low processor loads.

However, where an ASCE experiences flow control back pressure (see clause 8.5) – for example, when sending over slower links such as typical PSTN lines, or when sending over congested higher speed links – experience shows that significant performance enhancements may be achieved by the application of a dictionary-based compression scheme to selected ASPDUs – including already compressed data in UpdatePDU (orders) and UpdatePDU (bitmap) ASPDUs. This approach is termed general compression.

8.3.2.1 Legacy mode general compression

This Recommendation does not specify a general compression mechanism for legacy mode, but rather provides a compression scheme-independent mechanism whereby ASCEs can negotiate and use mutually agreed general compression schemes, as follows.

An ASCE shall negotiate the general compression scheme using the General.generalCompressionLevel and General.generalCompressionTypes capabilities. See clause 8.2.3 for further information on the general capability set. The GeneralCompressionLevel capability indicates which level of general compression is supported by an ASCE while the General.generalCompressionTypes capability is a set of bit flags, allowing an ASCE to indicate that it supports none, one or a selection of general compression schemes.

Note that the interpretation of the General.generalCompressedTypes capability is dependent on the negotiated General.generalCompressionLevel capability. Where the negotiated General.generalCompressionLevel capability is zero, then only the least significant bit flag of the General.generalCompressedTypes capability shall be considered. Where the negotiated generalCompressionLevel capability is greater than zero, then all bit flags within the General.generalCompressedTypes capability shall be considered.

Where an ASCE determines that other peer ASCEs can receive a particular general compression scheme, it may apply general compression to data ASPDUs on a selective basis – that is, an ASCE may apply general compression to none, some or all ASPDUs. The criteria by which an ASCE determines whether general compression is appropriate for a particular data ASPDU may include flow control status (see clause 8.5) and the size of the particular ASPDU. This Recommendation does not recommend a particular approach to the application of general compression – it is entirely a local matter for each ASCE.

Where an ASCE does not apply general compression to a data ASPDU, either because no suitable general compression scheme is available after capabilities negotiation, or where a suitable general compression scheme has been negotiated but the sending ASCE determines that general compression is inappropriate, the ASCE shall set the generalCompressedType to zero to indicate that the ASPDU is not general compressed, the uncompressedLength parameter to the ASPDU octet length and the generalCompressedLength parameter to zero. Where an ASCE applies general compression to a data ASPDU, it shall set the generalCompressedType to a value corresponding to one of the negotiated general compression schemes to indicate that the ASPDU is general compressed, the uncompressedLength parameter to the ASPDU octet length before compression and the generalCompressedLength parameter to the ASPDU octet length after compression. Setting both the uncompressedLength and generalCompressedLength parameters allows a receiving ASCE to check that the ASPDU length after decompression is consistent with its original uncompressed length.

Note that the interpretation of the generalCompressedType parameter in the ShareData header is dependent on the negotiated General.generalCompressionLevel capability. Where the negotiated General.generalCompressionLevel capability is zero, then only the most significant bit of the General.generalCompressedType parameter shall be considered. Where the negotiated General.generalCompressionLevel capability is greater than zero, then all bits of the General.generalCompressedType parameter shall be considered.

See Appendix I for informative values and information on legacy mode general compression.

8.3.2.2 Base mode general compression

This Recommendation specifies the optional use of [ITU-T V.42 *bis*] for general compression in base mode.

Where an ASCE determines that other peer ASCEs can receive ASPDUs general compressed using V.24 *bis*, it may apply V.42 *bis* general compression to data ASPDUs on a selective basis – that is, an ASCE may apply V.42 *bis* general compression to none, some or all ASPDUs. The criteria by which an ASCE determines whether V.42 *bis* general compression is appropriate for a particular data ASPDU may include flow control status (see clause 8.5) and the size of the particular ASPDU. This Recommendation does not recommend a particular approach to the application of V.42 *bis* general compression – it is entirely a local matter for each ASCE.

Where V.42 *bis* general compression is applied, the encoder is initialized during the sender's hosting synchronization and the decoder is initialized during the receivers' shadow synchronization (see clauses 8.6.2 and 8.6.3).

Where an ASCE does apply V.42 *bis* general compression to a particular data ASPDU, it shall set the generalCompressionSpecifier parameter to indicate that the ASPDU is V.42 *bis* general compressed. If the ASPDU is the first V.42 *bis* general compressed ASPDU after a V.42 *bis* encoder initialization, then the ASCE may also include the optional V.42 *bis* P1 and P2 parameters in the general CompressionSpecifier to allow peer ASCEs to optimize general compression-related resources.

8.4 ASCE activation

Within an AS session, an ASCE may be in one of the following three activation states:

- An inactive ASCE is not participating in the sharing of windows (i.e., it is not hosting windows, drawing shadow windows, or sending or receiving AS input). When an ASCE is inactive, it need not retain local resources that are required for hosting and/or sharing.
- A pending active ASCE is in the process of becoming active and is not participating in the sharing of windows (i.e., it is not hosting windows, drawing shadow windows, or sending or receiving AS input).
- An active ASCE is cooperating with other active ASCEs to share windows.

The following clauses describe ASCE activation and the handling of share identifiers for both legacy and base modes.

8.4.1 ASCE activation (legacy mode)

In the legacy mode of the AS protocol, ASCE activation and share identifier handling is largely independent of GCC and is performed using AS-specific protocol elements.

An ASCE joins the AS non-standard base session (see clause 7) using a GCC-Application-Enroll request with the active/inactive flag set to active and with limited capabilities. This does not make the ASCE active (with respect to the AS session states described above) but does ensure that the ASCE is visible to other ASCEs within the conference.

An ASCE may remain enrolled in the non-standard base session over a period, while moving from inactive to active state and back again a number of times. This distinction, between enrolment and activation, enables an ASCE to defer the commitment of application sharing-specific resources to those points within the conference where application sharing is actually taking place.

An ASCE that wishes to become active shall send either a DemandActivePDU or a RequestActivePDU to all ASCEs within the conference in the manner indicated in Table 6-3. The content of the DemandActivePDU is shown in Table 8-26 and the content of the RequestActivePDU is shown in Table 8-27.

It is recommended that an ASCE should initially send a RequestActivePDU to all ASCEs within the conference to determine if there are other active ASCEs in the AS session. If no active ASCEs respond to the RequestActivePDU, then, at some later point, when the ASCE wishes to start sharing, it should send a DemandActivePDU to all ASCEs within the conference to initiate activation.

Sending a DemandActivePDU requires that the sending ASCE shall have generated a unique share identifier, whereas RequestActivePDU does not. Sending DemandActivePDU may force a change in the current agreed share identifier. See clause 8.4.2 for further information on share identifiers.

On receipt of a DemandActivePDU, subject to successful capabilities negotiation (see clause 8.2), an ASCE may send ConfirmActivePDUs (see below) to all ASCEs within the conference. An inactive ASCE is not required to respond to a DemandActivePDU – it may choose to remain inactive. If an inactive ASCE responds to a DemandActivePDU, then it enters the active state.

If an ASCE is already active, then, on receipt of a RequestActivePDU and subject to successful capabilities negotiation (see clause 8.2), it shall send ConfirmActivePDUs (see below) to all ASCEs within the conference.

For both cases, the ConfirmActivePDU shall be sent to all ASCEs within the conference on all priority channels – high, medium and low – in the manner indicated in Table 6-3. The content of the ConfirmActivePDU is shown in Table 8-28. This ensures that any data on any channel is received after the ConfirmActivePDU. The first ConfirmActivePDU received on any channel causes the receiving pending inactive ASCE to enter the active state. Subsequent ConfirmActivePDUs shall be discarded.

Figure 3 illustrates the use of RequestActivePDU, DemandActivePDU and ConfirmActivePDU in the activation of three ASCEs, where initially no ASCEs are active and ASCE 1 uses RequestActivePDU to determine the activation state of other ASCEs within the conference. ASCE 1 then starts sharing and uses DemandActivePDU to initiate the activation of the other ASCEs within the conference.

See Figure 5 for a phase diagram summary of allowable transitions between ASCE states.

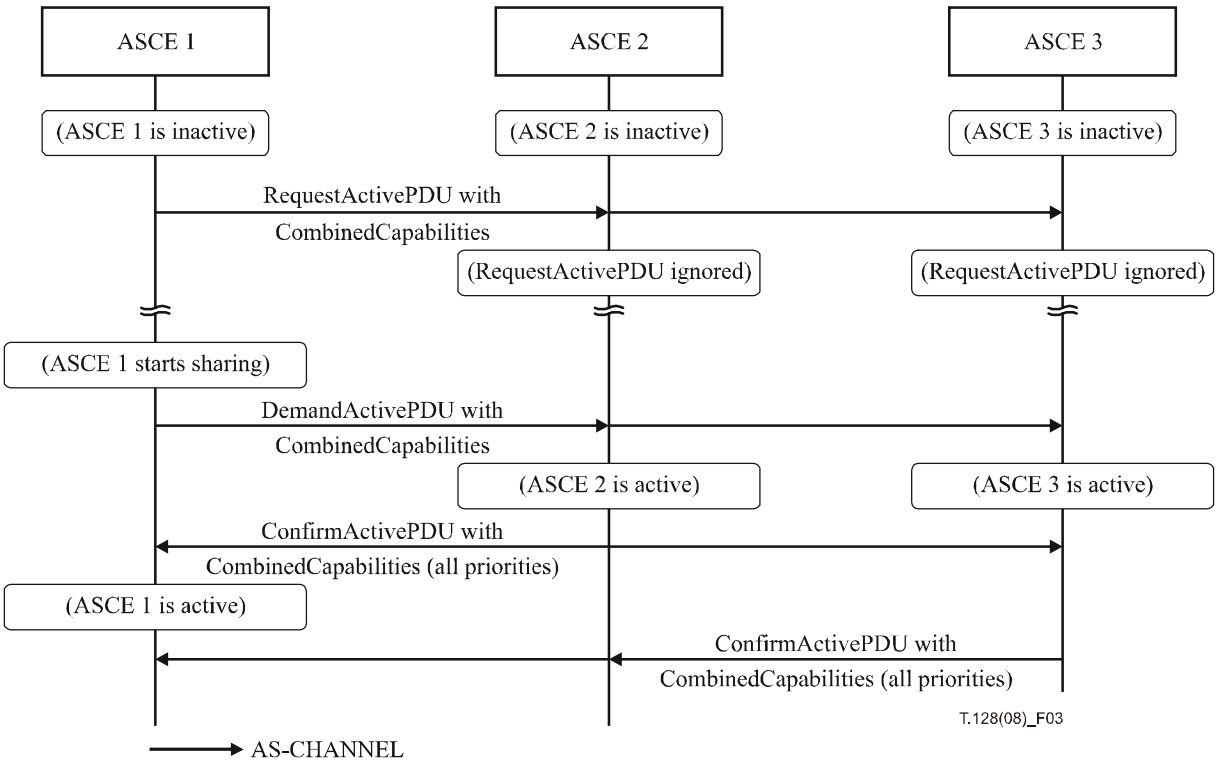


Figure 3 – Successfully activating three ASCEs (legacy mode)

Table 8-26 – DemandActivePDU

Parameter	Description
ShareControl header	The ShareControl header is described in clause 8.3.
shareID	This parameter is the proposed share identifier for use within the AS session. See clause 8.4.2 for a description of this parameter.
sourceDescriptor	This parameter is a null-terminated T.50 text string identifying this ASCE suitable for display to an end-user. This Recommendation does not place any interpretation upon the string contents.
combinedCapabilities	This parameter is a list of this ASCE's combined capabilities, which contains one copy of each of the legacy mode capability sets in any order. See clause 8.2 for further information on capabilities.

Table 8-27 – RequestActivePDU

Parameter	Description
ShareControl Header	The ShareControl header is described in clause 8.3.
sourceDescriptor	This parameter is a null-terminated T.50 text string identifying this ASCE suitable for display to an end-user. This Recommendation does not place any interpretation upon the string contents.
combinedCapabilities	This parameter is a list of this ASCE's combined capabilities, which contains one copy of each of the legacy mode capability sets in any order. See clause 8.2 for further information on capabilities.

Table 8-28 – ConfirmActivePDU

Parameter	Description
ShareControl header	The ShareControl header is described in clause 8.3.
shareID	This parameter is the current share identifier in use within the AS session. See clause 8.4.2 for a description of this parameter.
originatorID	This parameter is the MCS user ID of the ASCE that issued the DemandActivePDU or RequestActivePDU to which this is a response. This enables an ASCE receiving this ConfirmActivePDU to correlate it with a previous DemandActivePDU or RequestActivePDU.
sourceDescriptor	This parameter is a null-terminated T.50 text string identifying this ASCE suitable for display to an end-user. This Recommendation does not place any interpretation upon the string contents.
combinedCapabilities	This parameter is a list of this ASCE's combined capabilities, which contains one copy of each of the legacy mode capability sets in any order. See clause 8.2 for further information on capabilities.

To deactivate another active ASCE, an ASCE shall send a DeactivateOtherPDU to the ASCE that is to be deactivated in the manner indicated in Table 6-3. The content of the DeactivateOtherPDU is shown in Table 8-29. On receipt of a DeactivateOtherPDU, an ASCE shall become inactive. It is recommended that an ASCE should only issue DeactivateOtherPDU where it has received a DemandActivePDU, RequestActivePDU or ConfirmActivePDU with capabilities that would seriously affect the successful progress of application sharing within the conference. This situation should not occur where other ASCEs provide capabilities that conform to the legacy mode of the AS protocol.

An ASCE may deactivate all active ASCEs by sending a DeactivateAllPDU to all ASCEs within the conference in the manner indicated in Table 6-3. The content of the DeactivateAllPDU is shown in Table 8-31. On receipt of a DeactivateAllPDU, an ASCE shall become inactive. It is recommended that an ASCE should only issue DeactivateAllPDU in exceptional circumstances.

When an ASCE determines that there are no other active ASCEs within the AS session, it shall become inactive.

When an ASCE becomes inactive, it shall send a DeactivateSelfPDU to all ASCEs within the conference in the manner indicated in Table 6-3. The content of the DeactivateSelfPDU is shown in Table 8-30.

Figure 4 illustrates the use of DeactivateOtherPDU and DeactivateSelfPDU where ASCE 1 deactivates ASCE 3. On receipt of the DeactivateOtherPDU, ASCE 3 sends DeactivateSelfPDU to ASCEs 1 and 2, whereby both receive notification that ASCE 3 has become inactive.

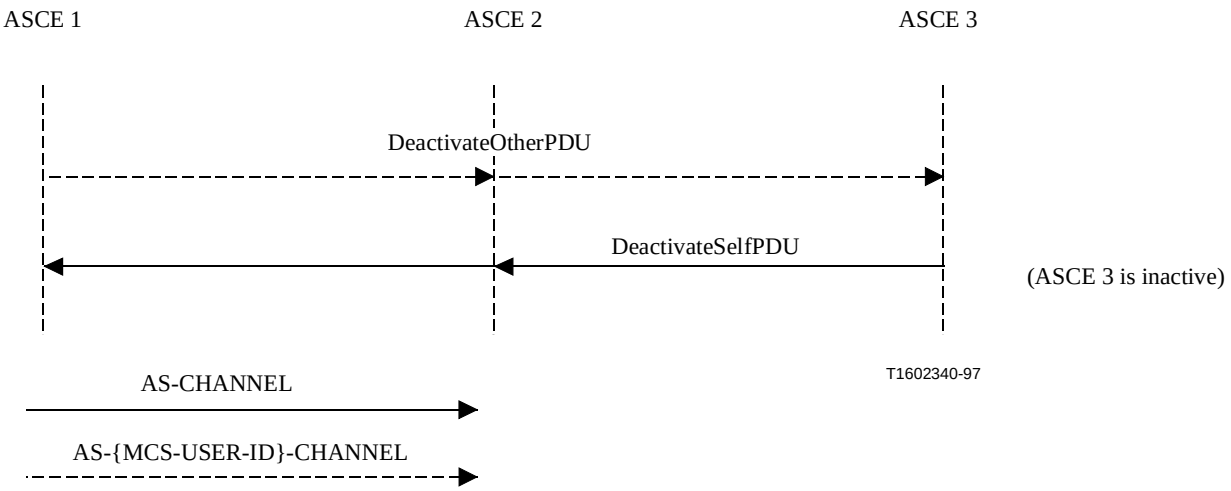


Figure 4 – Deactivating another ASCE

Table 8-29 – DeactivateOtherPDU

Parameter	Description
ShareControl header	The ShareControl header is described in clause 8.3.
shareID	This parameter is the current share identifier in use within the AS session. See clause 8.4.2 for a description of this parameter.
deactivateID	This parameter is the MCS user ID of the ASCE to be deactivated.
sourceDescriptor	This parameter is a null-terminated T.50 text string identifying this ASCE suitable for display to an end-user. This Recommendation does not place any interpretation upon the string contents.

Table 8-30 – DeactivateSelfPDU

Parameter	Description
ShareControl header	The ShareControl header is described in clause 8.3.
shareID	This parameter is the current share identifier in use within the AS session. See clause 8.4.2 for a description of this parameter.

Table 8-31 – DeactivateAllPDU

Parameter	Description
ShareControl header	The ShareControl header is described in clause 8.3.
shareID	This parameter is the current share identifier in use within the AS session. See clause 8.4.2 for a description of this parameter.
sourceDescriptor	This parameter is a null-terminated T.50 text string identifying this ASCE suitable for display to an end-user. This Recommendation does not place any interpretation upon the string contents.

8.4.2 Share identifiers (legacy mode)

A share identifier is a unique 32-bit handle present in (almost) all ASPDUs, which ASCEs use to detect and ignore "late" data generated with respect to previous activations and capabilities within the AS session. In the legacy mode of the AS protocol, share identifiers are constructed locally by

each ASCE and consist of the ASCE's MCS user ID (in the most significant 16 bits) and a monotonically increasing 2^{16} unsigned and wrapping counter (in the least significant 16 bits).

In the legacy mode of the AS protocol, the following ASPDUs do not contain share identifiers. All other legacy mode ASPDUs contain share identifiers.

- RequestActivePDU: See clause 8.4.1.
- FlowTestPDU: See clause 8.5.
- FlowResponsePDU: See clause 8.5.
- FlowStopPDU: See clause 8.5.

Each ASCE in an AS session shall maintain a local share identifier, which is its view of the current share identifier in use within the session, as follows:

- When an ASCE is inactive or has issued a RequestActivePDU, it shall set its local share identifier to the special value invalid.
- An ASCE shall generate a new proposed share identifier (by incrementing the counter part of the previous share identifier) when issuing a DemandActivePDU and shall set its local share identifier to that new identifier.
- An ASCE shall check the share identifier in an incoming DemandActivePDU or ConfirmActivePDU against its local share identifier and shall update its local share identifier as follows:
  - If its local share identifier is invalid, it shall set its local share identifier to the share identifier in the incoming ASPDU.
  - If its local share identifier is valid, it shall set its local share identifier to the higher value of the two share identifiers. Note that, because share identifiers are formed from a combination of MCS user ID and a local counter, the share identifier values cannot be equal.
- On receipt of a RequestActivePDU (when active), an ASCE shall set the share identifier value in its subsequent ConfirmActivePDU to its local share identifier.
- On issuing a DeactivateSelfPDU or DeactivateAllPDU, or on receipt of a DeactivateOtherPDU or DeactivateAllPDU, or when there are no other active ASCEs in the session, an ASCE shall set its local share identifier to the special value invalid.
- On receipt of all other ASPDUs containing a share identifier (see above), an ASCE shall discard the ASPDU where the share identifier value in the ASPDU does not match its local share identifier.

The net of the above is that ASCEs using the legacy mode of the AS protocol within an AS session cooperate to maintain a current agreed share identifier for all active ASCEs, with the current agreed share identifier potentially changing as new ASCEs become active.

Figure 5 presents a phase diagram summarizing the allowable transitions between ASCE states with respect to share identifier value. This shows two substates of the pending active state, depending on whether the ASCE has a proposed or invalid share identifier. It does not show ASPDUs and state/share identifier combinations that do not cause state transitions, nor does it show ASCE actions associated with state transitions.

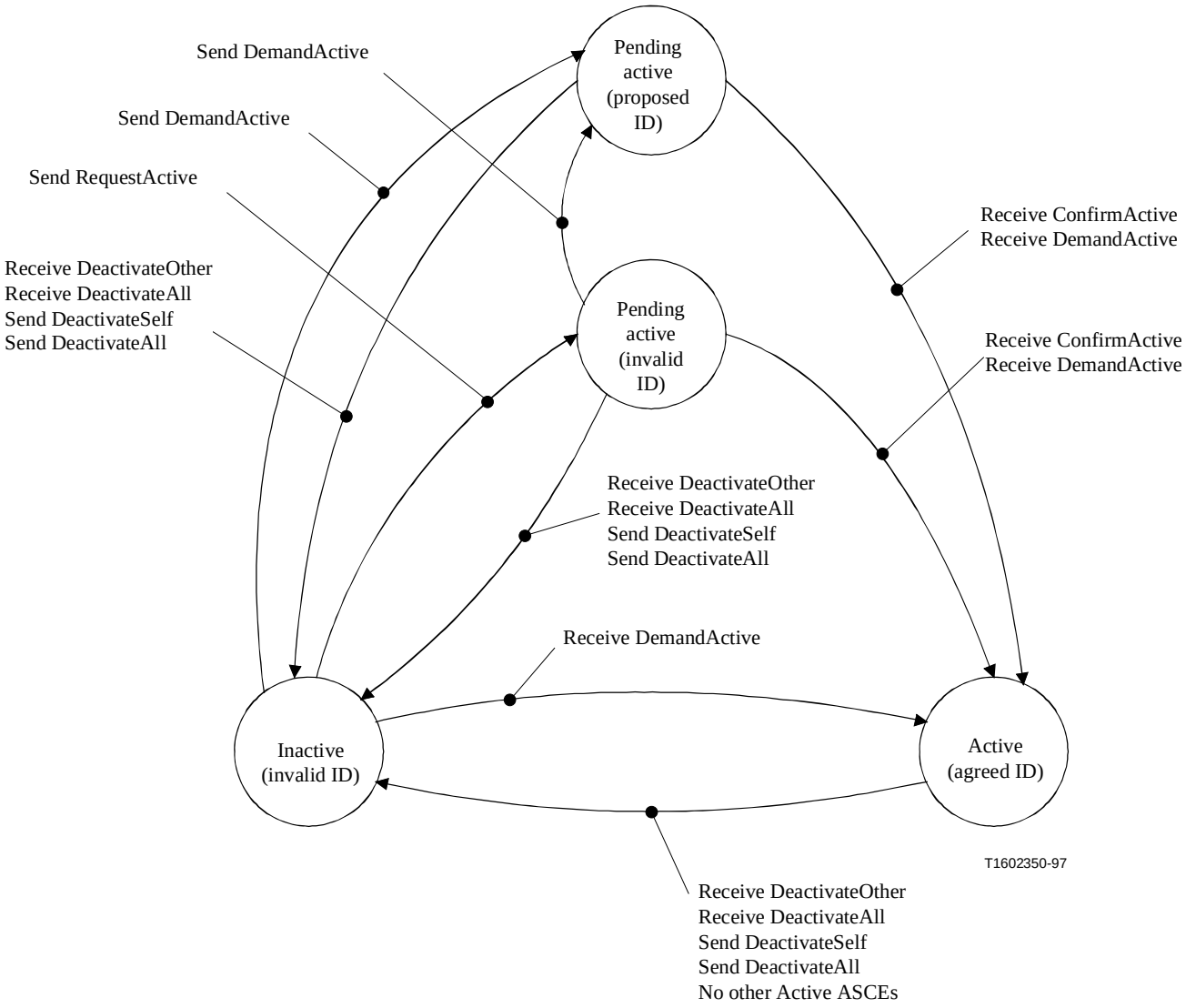


Figure 5 – ASCE state transitions and share identifier management (legacy mode)

8.4.3 ASCE activation and share identifiers (base mode)

In the base mode of the AS protocol, ASCE activation shall be performed according to [ITU-T T.121].

A share identifier is a unique 32-bit handle present in (almost) all ASPDUs, which ASCEs use to detect and ignore "late" AS data generated with respect to previous capabilities within the AS session. In the base mode of the AS protocol, an ASCE shall use the last roster instance number supplied by a GCC-Application-Roster-Report indication on the standard base session that contains both itself and other ASCEs enrolled active as its current share identifier.

In the base mode of the AS protocol, the following ASPDUs do not contain share identifiers. All other base mode ASPDUs contain share identifiers.

- FlowTestPDU: See clause 8.5.
- FlowResponsePDU: See clause 8.5.
- FlowStopPDU: See clause 8.5.

While the use of share identifiers allows an ASCE to avoid the use of "late" AS data, it is possible for an ASCE to receive "early" AS data that have been generated with respect to capabilities that have not yet been seen by the receiving ASCE. This occurs because GCC distributes roster

information via the GCC top provider while AS data are sent using (non-uniform) MCS-Send-Data primitives. This means that, for a simple example for a conference with two ASCEs (ASCEs A and B), where each of the ASCEs enrol active at about the same time, then:

- ASCE A may receive a roster report containing the capabilities for ASCEs A and B at time t_1;
- based on those capabilities, ASCE A starts sending AS data to ASCE B at time t_2;
- the AS data from ASCE A may arrive at ASCE B at time t_3; at this point, ASCE B has not received a roster report containing A;
- ASCE B may receive a roster report containing the capabilities for ASCE A at time t_4.

An ASCE can detect "early" AS data by the fact that the data's share identifier will be greater modulo 2^{16} than the last roster instance number supplied by a GCC-Application-Roster-Report indication on the standard base session. For this case, an ASCE shall process in advance or buffer such data until a GCC-Application-Roster-Report indication is received that identifies a roster instance number which is greater modulo 2^{16} than or equal to the one specified in the data. If an ASCE chooses to process such data in advance of the receipt of the proper GCC-Application-Roster-Report indication, it shall not assume that the capabilities corresponding to the data are within the bounds defined by its own capabilities, given that the roster change prompting the new application roster may have been triggered by the ASCE being either forcibly or voluntarily removed from the AS session.

8.5 Flow control

A key performance factor for application sharing is the ability to tailor the protocol content based on the potential traffic presented by local hosted window activity and the instantaneous aggregate bandwidth to other nodes to achieve maximum throughput and responsiveness. During the course of an AS session, the aggregate bandwidth available for application sharing will vary depending on a wide range of factors, including:

- the conference topology and link speeds;
- the activation status and behaviour of other ASCEs;
- the presence and/or behaviour of other APEs – such as T.126 and/or T.127 APEs;
- other services (such as video and/or audio) originating or terminating on nodes involved in the conference;
- the general traffic load (on shared, non-dedicated) networks.

As the instantaneous aggregate bandwidth available for application sharing is reduced, an ASCE may need to scale back the amount of data in flight in the network (e.g., by collapsing multiple overlapping screen updates, increasing compression ratios, etc.) such that, while the frequency of remote screen updates is reduced, overall responsiveness is maintained and, in particular, remote users on slower links do not "fall behind" the conference. As more instantaneous aggregate bandwidth is available for application sharing, an ASCE may be able to increase the amount of data in flight in the network, such that the frequency of remote screen updates and actual responsiveness is improved.

For an ASCE to effectively adjust its protocol content in this manner requires multipoint aggregate end-to-end flow control. Unfortunately, MCS does not define a suitable flow control mechanism. Therefore, this Recommendation defines an AS protocol-specific mechanism – AS flow control – whereby ASCEs can obtain information derived from the instantaneous aggregate bandwidth available to application sharing, which allows them to tailor their AS protocol stream accordingly.

AS flow control is applied on a per-channel and per-priority basis. A particular channel/priority combination is referred to as a flow. Flow control may be applied to none, some or all flows. This Recommendation only applies flow control to AS-CHANNEL flows and does not apply flow

control to AS-{MCS-USER-ID}-CHANNEL flows – that is, flow control is only applied to broadcast channel/priority combinations. This means that an ASCE can support up to three controlled flows per AS session.

For each controlled flow, an ASCE specifies a target round trip time and a target maximum amount of data in flight. While those targets are in force, an ASCE monitors the aggregate round trip time for that flow, via exchanges of FlowTestPDUs and corresponding FlowResponsePDUs (see below) and monitors the amount of data in flight, while seeking to maximize the amount of data in flight (up to the target maximum) and maintaining the target round trip time.

Where flow control is applied to a particular flow, an ASCE shall periodically send a FlowTestPDU for that flow to other ASCEs within the conference in the manner indicated in Table 6-3. The content of the FlowTestPDU is shown in Table 8-32. On receipt of a FlowTestPDU, an ASCE shall reply with a FlowResponsePDU to the issuing ASCE in the manner indicated in Table 6-3. The content of the FlowResponsePDU is shown in Table 8-33. The receipt of a FlowResponsePDU provides an estimate of the round trip delay to the particular responding ASCE. Note that this definition of round trip time includes all time spent in local, intermediate and final destination node queues and is therefore an estimate of end-to-end delivery time to the remote ASCE.

Table 8-32 – FlowTestPDU

Parameter	Description
flowID	This parameter identifies the controlled flow for the sending ASCE. The allowable values are in the range 0..127. This parameter is allocated by the sending ASCE and has no specific correlation with particular channels and/or priorities.
flowNumber	This parameter identifies this particular FlowTestPDU for this flowID (see above). The allowable values are in the range 0..255. An ASCE should increment this parameter for each successive FlowTestPDU, wrapping to 0 after using 255.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

Table 8-33 – FlowResponsePDU

Parameter	Description
flowID	This parameter identifies the controlled flow for the sending ASCE. The allowable values are in the range 0..127. The sending ASCE shall set this parameter to that used in the corresponding FlowTestPDU.
flowNumber	This parameter identifies this particular FlowResponsePDU on this flowID (see above). The allowable values are in the range 0..255. The sending ASCE shall set this parameter to that used in the corresponding FlowTestPDU.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

An ASCE is required to issue FlowTestPDUs periodically and is required to respond to FlowTestPDUs with FlowResponsePDUs – except where it has previously sent a FlowStopPDU (see below).

An ASCE should use the round trip times provided by the series of exchanges of FlowTestPDUs and FlowResponsePDUs to:

- maintain a dynamic estimate of the aggregate throughput for each controlled flow;
- adjust the sending of FlowTestPDUs and FlowResponsePDUs to reflect the estimate of the aggregate throughput;
- adjust the content of the outgoing AS protocol data stream to reflect the estimate of the aggregate throughput (see clause 8.5.2).

The implementation of efficient and predictable ASCE flow control is a major determinant in application sharing performance and usability. Therefore, this Recommendation requires that ASCEs implement a flow control algorithm either equivalent to, or superior to, the algorithm described in clause 8.5.1 below.

8.5.1 Flow control algorithm

The following flow control algorithm is described in terms of a set of flow control constants and variables, a set of specific operations on those variables and logic which describes under what conditions those operations are performed. Appendix I provides suggested constant values, initial, minima and maxima values for the flow control variables and suggested expressions for the operations, based on experience with a particular ASCE implementation. However, different ASCE implementations may find that different values and/or expressions deliver the required equivalent or superior flow control behaviour.

- An ASCE uses the following flow control constant values per controlled flow:
  - target_round_trip — This is the target round trip time for the flow.
  - target_in_flight — This is the target maximum number of octets that can be in flight for the flow.
  - max_queued_recv — This is the maximum number of received ASPDUs that can be locally queued for the flow before FlowResponsePDUs responses are deferred.

  Appendix I provides suggested values for these flow control constants based on experience with a particular ASCE implementation.
- An ASCE maintains the following flow control variables per controlled flow:
  - max_in_flight — This is the current maximum number of octets that can be in flight for the flow.
  - flow_period — This is the current minimum period between sending FlowTestPDUs.

  These variable values are adjusted dynamically (see below) based on the data sent by the ASCE and on the observed round trip delay to other ASCEs.

  Appendix I provides suggested initial values and minima and maxima for these flow control variables based on experience with a particular ASCE implementation.
- An ASCE may perform the following operations on the flow control variables:
  - Decrease max_in_flight;
  - Increase max_in_flight;
  - Decrease flow_period;
  - Increase flow_period;

  Appendix I provides suggested expressions for these operations based on experience with a particular ASCE implementation.

- When sending data for a controlled flow, an ASCE should check whether this data results in the number of octets in flight exceeding max_in_flight.
  - Where that is not the case, the ASCE should send the data.
  - Where that is the case, the ASCE should not send the data (i.e., the flow control algorithm is applying local back pressure within the ASCE).

  How back pressure is applied within the ASCE is a local matter.
- On receipt of a FlowTestPDU, an ASCE should check whether it is still processing more than max_queued_recv ASPDUs on the flow.
  - Where that is not the case, it should respond with a FlowResponsePDU referencing that flow and using the flowNumber supplied in the FlowTestPDU.
  - Where that is the case, it should pend the FlowResponsePDU until the local backlog has been reduced – i.e., until the number of locally queued ASPDUs on the flow is less than or equal to max_queued_recv.

  The check for a backlog of previously received data avoids problems where a receiving ASCE is the slowest element in the end-to-end network, where an unconditional FlowResponsePDU would result in an accumulation of unprocessed data at the receiver.
- On receipt of a FlowResponsePDU, an ASCE should check whether the flow control algorithm is applying local back pressure.
  - Where that is not the case, no action is necessary.
  - Where that is the case and the round trip delay is worse than the target_round_trip, it should decrease max_in_flight and increase flow_period. This will have the effect of reducing throughput by encouraging earlier back pressure.
  - Where that is the case and the round trip delay is better than the target_round_trip, it should increase max_in_flight. This will have the effect of lifting back pressure and increasing throughput when data is next sent (provided a worse round trip delay does not happen in the interim). If the new max_in_flight has reached target_in_flight, then the ASCE should also decrease flow_period. This will have the effect of increasing the algorithm's sensitivity to any deterioration in round trip times as a result of the increased data in flight.

  This means that the ASCE will only adjust max_in_flight or flow_period where local back pressure is being applied. This is because this algorithm uses the observed round trip as an estimate of throughput, which is only a reasonable correlation where the round trip coincides with a period of busy data traffic – i.e., when this ASCE is applying local back pressure because there are at least max_in_flight octets in flight.

  Note that an ASCE should take care that successive estimates of round trip during local back pressure do not result in oscillation – for example, where successive round trips within the same flow_period are worse than the target (resulting in a decrease in max_in_flight and increase in flow_period) and then better than the target (resulting in an increase in max_in_flight and possible decrease in flow_period). A recommended heuristic approach is to bias the decision such that a reduction in throughput is effected for cases where the round trip is simply worse than the target_round_trip, whereas an increase in throughput is only affected for cases where the round trip is less than half of the target_round_trip.

  The ASCE should apply this logic for each received FlowResponsePDU within a flow_period – with the early responders setting initial values for the period and later responders adjusting those values to reflect the bandwidth characteristics applicable to the corresponding ASCEs.

The flow_period is the current minimum time between FlowTestPDUs. For example, if the flow_period is five milliseconds, then if an ASCE sends a FlowTestPDU at the start of a

flow_period and receives FlowResponsePDUs from all responding ASCEs (see below) within four milliseconds, then it should not send the next FlowTestPDU for a further millisecond. Conversely, if it has not received FlowResponsePDUs for all responding ASCEs within five milliseconds, it should not send the next FlowTestPDU until it has received FlowResponsePDUs for all responding ASCEs or some larger upper time bound has been reached. This approach means that, for a particular flow_period, the algorithm incrementally responds to the observed round trip for all responding ASCEs – provided they respond within a reasonable time.

The group of responding ASCEs for which responses are expected within a flow_period may not necessarily correspond to the group of all other active ASCEs. A sending ASCE should only include an ASCE within its group of responding ASCEs when that ASCE has provided an initial response and should exclude ASCEs that do not respond within the upper time bound (at least, until such ASCEs respond again within subsequent flow_periods). This approach ensures that faulty ASCEs do not skew the algorithm.

The net of the above is that sending ASCEs will (for a particular flow):

- apply local back pressure when the amount of data in flight reaches the current allowable data in flight;
- increase or decrease the allowable data in flight to reflect the aggregate instantaneous bandwidth, based on the observed round trip delay;
- increase or decrease the flow_period (and thus the proportion of bandwidth occupied by flow ASPDUs) to reflect the aggregate instantaneous bandwidth.

An ASCE may remove itself from other ASCEs' flow control responding groups by sending a FlowStopPDU for that flow to all other ASCEs within the conference in the manner indicated in Table 6-3. The content of the FlowStopPDU is shown in Table 8-34. This mechanism is primarily of use in the legacy mode of the AS protocol, as it allows an ASCE to remain enrolled in the AS-CHANNEL (and therefore still receiving FlowTestPDUs) but to be inactive (in the sense of ASCE activation – see clause 8.4) and ensures that its failure to respond does not skew other active ASCEs' flow control calculations.

When the ASCE again wishes to be included in flow control calculations, it should respond to the next FlowTestPDU with a FlowResponsePDU, which will have the effect of adding it back into other ASCEs' flow control responding groups for that flow.

Table 8-34 – FlowStopPDU

Parameter	Description
flowID	This parameter identifies the controlled flow for the sending ASCE. The allowable values are in the range 0..127. The sending ASCE shall set this parameter to that used in the corresponding FlowTestPDU.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.5.2 Response to back pressure

When back pressure is being applied within the ASCE as a result of the flow control algorithm described in clause 8.5.1, an ASCE may process pending AS data such that the volume of the data is reduced.

Example strategies for output AS data include the following:

- Switching between sending orders and bitmap data:

Where a local application is very actively drawing into a hosted window and the ASCE is experiencing flow control back pressure, the ASCE may prefer to accumulate bounds information for the drawing activity, rather than sending the orders, and then subsequently send bitmap data for the accumulated bounds. This may reduce the frequency of updates at remote ASCEs, but will minimize the data in flight and ensures that remote ASCEs "keep up" with the drawing activity. The switching points between orders and bitmap data and vice versa will depend on application drawing behaviour, flow control and the particular ASCE implementation.

- Collapsing overlapping updates – this process is referred to as spoiling:

  Where the ASCE is experiencing flow control back pressure and the local application draws a sequence of overlapping or occluding order or bitmap data updates, then the sequence may (in some cases) be collapsed into a smaller sequence and/or a single update. Again, this may reduce the frequency of updates at remote ASCEs, but will minimize the data in flight and ensure that remote ASCEs "keep up" with the drawing activity.

Where an ASCE is experiencing flow control back pressure and there is a high level of end-user pointing device activity, the ASCE may collapse a sequence of pointing device move events into a single move – this process is referred to as input spoiling. Input spoiling may also be applied at the hosting ASCE, where a sequence of pointing device events is queued within the ASCE awaiting injection into the local terminal environment. Both send and receive input spoiling may reduce the frequency of pointing device move updates presented to hosted windows and therefore degrade the smoothness of pointing device movement, but will minimize the data in flight and ensure that remote ASCEs "keep up" with the pointing device activity.

This Recommendation does not specify how an ASCE should adjust the content of either the output or input AS protocol data streams when back pressure is in effect, but does require that the ASCE shall ensure that the protocol stream contains sufficient information to allow peer ASCEs to correctly display shadow windows and/or correctly control hosted applications. Similarly, where the output stream relies on control information (such as window list or palette information) that has changed on the hosting ASCE, it is the hosting ASCE's responsibility to ensure that the control information is sent prior to the output. All order-dependent control and output ASPDUs are sent at low priority, so that the sending ASCE can reliably order such output.

8.6 Synchronization

When an ASCE is active within an AS session, there are several classes of session events that may require it to update protocol-related resources and/or state, or to send ASPDUs which allow other ASCEs to perform complementary changes. The ASCE processing associated with these session events is referred to as synchronization.

The AS protocol defines four classes of synchronization – referred to as ASCE, hosting, shadow and input synchronization. Each synchronization class defines a related set of ASCE synchronization requirements associated with a particular range of session events. For example, hosting synchronization defines a set of synchronization requirements that apply to an ASCE that is hosting (or starting or stopping hosting) windows.

This clause provides a summary of those ASCE synchronization operations required to support the AS protocol. It does not describe each operation in detail, but rather references subclauses elsewhere in this Recommendation that describe the relevant functional area.

Where the description references a particular local resource, such as an identifier value or a sending cache, it does so in logical terms, which do not assume a particular local ASCE implementation – the only requirement is that the synchronizing ASCE (subsequently) achieves the required protocol effect. In addition, it does not address potential optimizations that may arise from collapsing a series of repeated synchronization operations.

8.6.1 ASCE synchronization

When an ASCE becomes active, it shall perform the following synchronization operations. See clause 8.4 for further information on ASCE activation and deactivation.

- It shall set its window list identifier sequence number part to zero. See clause 8.10 for further information.
- It shall set its window activation identifier to zero. See clause 8.11 for further information.
- It shall set its control identifier to zero. See clause 8.12 for further information.

The AS protocol does not require any specific synchronization when an ASCE becomes inactive.

When an active ASCE detects that another ASCE has become active, it shall perform the following synchronization operations:

- It shall determine the negotiated capabilities for all (i.e., existing and new) active ASCEs within the AS session and use these capabilities as the basis for constructing all subsequent data ASPDUs. See clause 8.2 for further information on capabilities and capabilities negotiation.
- Before sending any other data ASPDUs, it shall send a SynchronizePDU on all outgoing streams/priorities (see clause 8.3.1) to all ASCEs within the conference in the manner indicated in Table 6-3 and shall mark all incoming streams/priorities for each other active ASCE as pending synchronization. Each incoming stream/priority for each other active ASCE remains pending synchronization until it receives a SynchronizePDU – and until that point discards all incoming data ASPDUs on that stream/priority from the particular ASCE. This ensures that, where the share identifier does not change as a result of the new ASCE becoming active (see clause 8.4.2), any data arriving from other ASCEs relating to the previous negotiated combined capabilities is discarded. A particular SynchronizePDU is directed on a single stream/priority for a particular other active ASCE, but is sent on the AS-CHANNEL (and therefore to all ASCEs). Each SynchronizePDU includes the MCS user ID of the intended target ASCE, so that it can be discarded by ASCEs that are not the intended target. This mechanism ensures that the SynchronizePDU arrives before any broadcast data for the target ASCE. The content of the SynchronizePDU is shown in Table 8-35.
- It shall discard its list of matched fonts, reset its count of received FontPDUs and may send a FontPDU to re-advertise its set of matchable fonts. See clause 8.8 for further information.
- It shall send either a cooperate or detach ControlPDU to advertise its control state. If it is cooperating and it has the control identifier, it also shall send a grant control ControlPDU referencing itself to all active ASCEs to advertise that it owns the control identifier. See clause 8.12 for further information.
- If the ASCE is hosting windows, it shall also perform the relevant hosting synchronization operations (see clause 8.6.2 below).
- It shall perform the relevant input synchronization operations (see clause 8.6.4 below).

ASCE synchronization as a result of other ASCEs becoming active may generate significant local processing load and significant ASPDU traffic, especially where it also initiates host synchronization. It is recommended that where an ASCE detects that a number of other ASCEs have become active in the conference in close succession, it implements ASCE synchronization so that it defers some or all of the required synchronization for each newly-active ASCE until a significant proportion or all of the ASCEs are active, with the proviso that the deferment is time limited in proportion to the number of ASCEs in the group, and some arbitrary maximum, and that responsiveness to activation is maintained.

When another ASCE becomes inactive, an ASCE may be required to perform the following synchronization operations. See clause 8.4 for further information on ASCE activation and deactivation.

- Where the other ASCE becoming inactive, results in the local ASCE becoming inactive (because it is the only active ASCE remaining within the AS session), then no ASCE synchronization is required.
- Where the ASCE that became inactive owned the control identifier, then each remaining active ASCE is required to participate in an exchange of ControlPDUs to re-establish a single ASCE as the control identifier owner. See clause 8.12.1 for further information.
- Where other ASCEs remain active, the local ASCE may redetermine the negotiated capabilities (see clause 8.2) for all remaining active ASCEs within the AS session and use these capabilities as the basis for constructing all subsequent data ASPDUs. This is optional, as the negotiated capabilities for the previously active ASCEs were negotiated inclusive of the remaining ASCEs, and while the renegotiated capabilities may allow for the construction of a more optimal AS protocol stream, the previously negotiated capabilities are still valid. See clause 8.2 for further information.
- Where other ASCEs remain active, the local ASCE may also rematch its list of matched fonts based on the font information received from the remaining active ASCEs. Again, this is optional and while it may result in additional matched fonts, the previously matched fonts are still valid. See clause 8.8 for further information.

Table 8-35 – SynchronizePDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
targetUser	This parameter is the MCS user ID of the ASCE to which this ASPDU is directed. Where an ASCE receives a SynchronizePDU containing a MCS user ID other than its own, it shall discard it.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.6.2 Hosting synchronization

Hosting synchronization is the set of synchronization operations that are associated with the hosting of windows.

When an ASCE first starts hosting windows, when it is hosting windows and a new ASCE becomes active, or when it is hosting windows and a capability set is updated (see clause 8.2), it shall perform the following synchronization operations:

- It shall send an UpdatePDU (synchronization) to all ASCEs within the conference in the manner indicated in Table 6-3. The UpdatePDU (synchronization) shall be sent before any other ASPDUs generated by the sending ASCE as a result of hosting synchronization and notifies other ASCEs that this ASCE is hosting windows. On receipt of an UpdatePDU (synchronize), an ASCE shall perform any relevant shadow synchronization (see clause 8.6.3 below). The content of the UpdatePDU (synchronization) is shown in Table 8-36.
- It shall reset its sending colour pointer cache. See clause 8.14 for further information.
- In the base mode of the AS protocol only, it shall reset its sending colour space to RGB with no colour accuracy information.

- It shall pend the sending of an UpdatePDU (palette) with a new palette, which shall be sent before any subsequent bitmap data.
- It shall reset its order encoding state. See clause 8.16.3 for further information.
- It shall reset its sending bitmap cache. See clause 8.16.7 for further information.
- It shall reset its sending ColorTable cache. See clause 8.16.8 for further information.
- It shall reset its sending desktop save cache. See clause 8.16.17 for further information.
- It shall send an ApplicationPDU with action NotifyHostedApplications indicating the number of hosted applications on the local terminal.
- It shall send a WindowListPDU containing information on the current local window structure, which shall be sent before any WindowActivation. See clause 8.10 for further information.
- It shall send a WindowActivationPDU providing information on the local window activation status. See clause 8.11 for further information.
- It shall queue a desktop origin order, which shall be sent in an UpdatePDU (orders) ASPDU before any subsequent bitmap data or orders. See clause 8.16.18 for further information.
- It shall construct an AS output stream, consisting of a mixture of bitmap data and/or orders, containing sufficient information to allow other ASCEs to draw shadow windows corresponding to its hosted windows.

When an ASCE stops hosting windows, but remains active, it shall send an ApplicationPDU with action NotifyHostedApplications indicating that zero applications are now hosted, to allow other ASCEs to perform any required shadow synchronization.

Table 8-36 – UpdatePDU (synchronization)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.6.3 Shadow synchronization

Shadow synchronization is the set of synchronization operations that are associated with the drawing of shadow windows. It is performed with respect to one other or all other hosting ASCEs.

When an ASCE receives an UpdatePDU (synchronization) from another ASCE (see clause 8.6.2), it is required to perform shadow synchronization with respect to that hosting ASCE. Where an ASCE determines that a capability set has been updated (in legacy mode, on receipt of an UpdateCapabilityPDU; in base mode, on receipt of a GCC-Application-Roster-Report indication with changed capabilities), it is required to perform shadow synchronization with respect to all hosting ASCEs. Where shadow synchronization is required, an ASCE shall perform the following synchronization operations for each affected hosting ASCE:

- It shall reset its receiving colour pointer cache for that hosting ASCE. See clause 8.14 for further information.
- In the base mode of the AS protocol only, it shall reset its receiving colour space for that hosting ASCE to RGB with no colour accuracy information.
- It shall reset its order decoding state for that hosting ASCE. See clause 8.16.3 for further information.

- It shall reset its receiving bitmap cache for that hosting ASCE. See clause 8.16.7 for further information.
- It shall reset its receiving ColorTable cache for that hosting ASCE. See clause 8.16.8 for further information.
- It shall reset its receiving desktop save cache for that hosting ASCE. See clause 8.16.17 for further information.
- It shall reset its desktop origin to (0,0) for that hosting ASCE. See clause 8.16.18 for further information.

The AS protocol does not require any specific synchronization when an ASCE detects that another ASCE is no longer hosting – for example, when a hosting ASCE becomes inactive or the ASCE receives an ApplicationPDU with action NotifyHostedApplications indicating that zero applications are hosted on a (previously) hosting ASCE. However, an ASCE may use the receipt of an ApplicationPDU with action NotifyHostedApplications and zero applications to free local resources (such as receive caches) that were allocated with respect to that ASCE.

8.6.4 Input synchronization

Input synchronization is the set of synchronization operations that are associated with the maintenance of keyboard state between controlling and controlled ASCEs. See clause 8.18 for further information.

When an ASCE detects that a new ASCE has become active, it shall reset its sending keyboard state and queue an input synchronization event for the next InputPDU. On receipt of an input synchronization event within an InputPDU, an ASCE shall reset its receiving keyboard state for the issuing ASCE.

8.7 Remote sharing

In some application sharing scenarios (e.g., remote working and/or help desk support), it is useful for sharing of applications and/or windows to be initiated remotely – that is, applications running on the local terminal are shared not by a local end-user or programmatic action, but rather as a result of a request from a peer ASCE. This is referred to as remote sharing.

When an ASCE wishes to initiate remote sharing on a peer ASCE, it shall send a RemoteSharePDU with action request remote share and an encrypted password to the peer ASCE in the manner described in Table 6-3. The content of the RemoteSharePDU is shown in Table 8-37.

On receipt of a request remote share RemoteSharePDU, an ASCE utilizes a purely local mechanism (such as interacting with the local end-user) to determine whether to allow remote sharing from the requesting ASCE and/or to validate the supplied password. If it determines to accept the remote share request, it responds with a confirm remote share RemoteSharePDU to the requesting ASCE and shares those local applications and/or windows that are eligible for remote sharing. The particular list of local applications and/or windows that are eligible for remote sharing is a local matter. For example, an ASCE may offer configuration options, whereby end-users can restrict the list of remote sharable applications and/or windows by class, by name or by requesting ASCE.

If the ASCE determines not to accept the remote share request, it responds with a deny remote share RemoteSharePDU to the requesting ASCE, indicating why the remote share attempt was denied. Values are defined for the following denial cases:

- The requesting ASCE supplied an invalid password.
- The receiving ASCE does not support remote sharing or does not have remote sharing enabled.

- The receiving ASCE has already executed a remote share. That is, its defined remote sharable applications and/or windows are already shared.
- The receiving ASCE has requested a remote share to a third ASCE.

Table 8-37 – RemoteSharePDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
action	This parameter identifies the particular RemoteSharePDU action. The allowable values are as follows: • Request remote share. • Confirm remote share. • Deny remote share.
additionalData	This parameter provides optional additional data for specific actions: • For the request remote share action, this parameter shall be the MCS user ID of the sending ASCE. This allows receiving ASCEs to implement remote share security on a per-remote ASCE basis. • For the deny remote share action, this parameter provides a reason code for the denial, as follows: – Incorrect password. – Remote share not enabled/supported. – Remote share in operation (incoming). – Remote share in operation (outgoing).
encryptedPassword	This parameter is only present for a request remote share message, where it is an X.509 protected simple authentication password without optional timestamp or optional random number.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.8 Fonts

An ASCE sends font information to all ASCEs within the conference by sending a FontPDU in the manner described in Table 6-3. The content of the FontPDU is shown in Table 8-38.

Font attribute information provides information on those fonts meeting the requirements of the AS protocol for which the sending ASCE is prepared to receive text and extended text orders (see clause 8.16.11 and clause 8.16.12).

An ASCE may send a FontPDU during ASCE synchronization to allow other active ASCEs to perform font matching, which is a prerequisite for the sending of text and extended text orders within the AS session. See clause 8.6 for further information on synchronization. An ASCE need not send FontPDUs where it does not support allowable local fonts (see clause 8.8.1) and/or where it does not support text and extended text orders.

Where an ASCE does not match a particular font and that font is then used locally by an application executing on the local terminal that is shared into the conference, the ASCE cannot send a text or extended text order referencing that font. However, it can send an UpdatePDU (bitmap) ASPDU containing the bits from the local terminal display area corresponding to the text bounds, which should result in an equivalent shadow window visual appearance on peer ASCEs.

For example, the current ASCE protocol only allows advertising of fonts where some or all of the codepoints have glyphs in the full or core AS protocol code page (see clause 8.8.1) and the text and extended text order codepoint lists assume a single-byte code page. This precludes the advertising of double-byte fonts or the sending of text or extended text orders for such fonts. However, where such fonts are in use for text drawing within shared applications at one or more ASCEs within a conference, the ASCEs can use suitable UpdatePDU (bitmap) ASPDUs to achieve the same shadow window visual appearance on peer ASCEs.

Table 8-38 – FontPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
entrySize	This parameter specifies the size in octets of a font attribute in the fontList parameter (see below). This parameter is only allowable for legacy mode FontPDUs.
fontList	This parameter is a list of font attributes.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

A receiving ASCE shall treat the position of font attributes in the FontPDU as a zero-offset numeric identifier, referred to as the FontID, for subsequent use during font matching and when sending text and extended text orders. For example, where a FontPDU contains 6 font attributes, the receiving ASCE shall use FontIDs 0..5 respectively to refer to the corresponding fonts at the sending ASCE. See Table 8-39.

Table 8-39 – Font attributes

Parameter	Description
faceName	This parameter is a null-terminated T.50 text string specifying the face name of the font.
fontFlags	This parameter is a set of bit flags indicating the characteristics of the font. Defined bit flag values are as follows. • Fixed pitch If this flag is set, it indicates that the font is fixed pitch. If it is not set, the font is proportional. • Fixed size If this flag is set, it indicates that the font is fixed size. If it is not set, the font is scalable.
averageWidth	This parameter is the average character width in pixels of the font. This is the average of the character widths for codepoints 0x61 through 0x7A plus codepoint 0x20 (i.e., "a" through "z" plus "space" for the AS protocol code page – see clause 8.8.1), where the character width is the character cell for fixed pitch fonts and the sum of the A + B + C widths for proportional fonts. Where the font is scalable, this value shall be calculated for a font of size 100 × 100 pixels.
height	This parameter is the character cell height in pixels for this font. Where the font is scalable, this value shall be calculated for a font of size 100 × 100 pixels.
aspectX	This parameter specifies the horizontal aspect in pixels per inch of the device for which the font was designed.

Table 8-39 – Font attributes

Parameter	Description
aspectY	This parameter specifies the vertical aspect in pixels per inch of the device for which the font was designed.
signature1	This parameter is the signature checksum for the first codepoint group. It is calculated as the sum of the character widths for codepoints 0x30 through 0x5A plus 0x24 through 0x26 inclusive divided by 2 and truncated to 8 bits, where the character width is the character cell for fixed pitch fonts and the sum of the A + B + C widths for proportional fonts. Where the font is scalable, this value shall be calculated for a font of size 100 × 100 pixels. See clause 8.8.1 for further information on codepoints and code pages. The combination of signature1 equals zero, signature2 equals zero and signature3 equals zero represents the special value NO_SIGNATURE.
signature2	This parameter is the signature checksum for the second codepoint group. It is calculated as the sum of the character widths for codepoints 0x20 through 0x7E inclusive, less the sum of the character widths for the codepoints specified for calculation for the signature1 parameter, with the result divided by 2 and truncated to 8 bits, where the character width is the character cell for fixed pitch fonts and the sum of the A + B + C widths for proportional fonts. Where the font is scalable, this value shall be calculated for a font of size 100 × 100 pixels. See clause 8.8.1 for further information on codepoints and code pages. The combination of signature1 equals zero, signature2 equals zero and signature3 equals zero represents the special value NO_SIGNATURE.
signature3	This parameter is the signature checksum for the third codepoint group. It is calculated as the sum of the character widths for codepoints 0x00 to 0x1E plus 0x80 through 0xFF inclusive. Where the font is scalable, this value shall be calculated for a font of size 100 × 100 pixels. See clause 8.8.1 for further information on codepoints and code pages. If the calculated values of the signature1, signature2 and signature3 parameters are all zero, then signature3 shall be set to zero. The combination of signature1 equals zero, signature2 equals zero and signature3 equals zero represents the special value NO_SIGNATURE.
codePage	This parameter indicates which AS protocol code page codepoints are supported by this font. Allowable values are: • All defined codepoints. • Core codepoints only. See clause 8.8.1 for further information on code points and code pages.
nonStandardFontAttributes	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard font attributes allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.8.1 Code page

The AS protocol code page consists of the ISO/IEC 8859-1 (Latin-1) code page with additional codepoints in the ranges 0x82-0x8C, 0x91-0x9C plus 0x9F. Codepoints in the ranges 0x00 through 0x1F, 0x7F through 0x81, 0x8D through 0x90, and codepoints 0x9D and 0x9E are not in [ISO/IEC 8859-1] and are not AS extensions – and are therefore not in the AS protocol code page.

Codepoints in the range 0x20 through 0x7E are considered to be the core AS protocol code page codepoints.

The AS protocol code page break character is codepoint 0x20.

An ASCE shall only include font attributes in a FontPDU for fonts where all codepoints have glyphs in the AS protocol code page. Similarly, it shall only send codepoints in text or extended text orders where the corresponding glyphs are present in the AS protocol code page.

Table 8-40 summarizes the supported codepoints in the AS protocol code page, indicating whether the codepoint is in [ISO/IEC 8859-1] or is an AS extension, and itemizes the corresponding Unicode [ISO/IEC 10646] code and name for each codepoint.

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x0020	√	0x0020	SPACE
0x0021	√	0x0021	EXCLAMATION MARK
0x0022	√	0x0022	QUOTATION MARK
0x0023	√	0x0023	NUMBER SIGN
0x0024	√	0x0024	DOLLAR SIGN
0x0025	√	0x0025	PERCENT SIGN
0x0026	√	0x0026	AMPERSAND
0x0027	√	0x0027	APOSTROPHE
0x0028	√	0x0028	LEFT PARENTHESIS
0x0029	√	0x0029	RIGHT PARENTHESIS
0x002A	√	0x002A	ASTERISK
0x002B	√	0x002B	PLUS SIGN
0x002C	√	0x002C	COMMA
0x002D	√	0x002D	HYPHEN-MINUS
0x002E	√	0x002E	FULL STOP
0x002F	√	0x002F	SOLIDUS
0x0030	√	0x0030	DIGIT ZERO
0x0031	√	0x0031	DIGIT ONE
0x0032	√	0x0032	DIGIT TWO
0x0033	√	0x0033	DIGIT THREE
0x0034	√	0x0034	DIGIT FOUR
0x0035	√	0x0035	DIGIT FIVE
0x0036	√	0x0036	DIGIT SIX
0x0037	√	0x0037	DIGIT SEVEN
0x0038	√	0x0038	DIGIT EIGHT
0x0039	√	0x0039	DIGIT NINE
0x003A	√	0x003A	COLON
0x003B	√	0x003B	SEMICOLON

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x003C	√	0x003C	LESS-THAN SIGN
0x003D	√	0x003D	EQUALS SIGN
0x003E	√	0x003E	GREATER-THAN SIGN
0x003F	√	0x003F	QUESTION MARK
0x0040	√	0x0040	COMMERCIAL AT
0x0041	√	0x0041	LATIN CAPITAL LETTER A
0x0042	√	0x0042	LATIN CAPITAL LETTER B
0x0043	√	0x0043	LATIN CAPITAL LETTER C
0x0044	√	0x0044	LATIN CAPITAL LETTER D
0x0045	√	0x0045	LATIN CAPITAL LETTER E
0x0046	√	0x0046	LATIN CAPITAL LETTER F
0x0047	√	0x0047	LATIN CAPITAL LETTER G
0x0048	√	0x0048	LATIN CAPITAL LETTER H
0x0049	√	0x0049	LATIN CAPITAL LETTER I
0x004A	√	0x004A	LATIN CAPITAL LETTER J
0x004B	√	0x004B	LATIN CAPITAL LETTER K
0x004C	√	0x004C	LATIN CAPITAL LETTER L
0x004D	√	0x004D	LATIN CAPITAL LETTER M
0x004E	√	0x004E	LATIN CAPITAL LETTER N
0x004F	√	0x004F	LATIN CAPITAL LETTER O
0x0050	√	0x0050	LATIN CAPITAL LETTER P
0x0051	√	0x0051	LATIN CAPITAL LETTER Q
0x0052	√	0x0052	LATIN CAPITAL LETTER R
0x0053	√	0x0053	LATIN CAPITAL LETTER S
0x0054	√	0x0054	LATIN CAPITAL LETTER T
0x0055	√	0x0055	LATIN CAPITAL LETTER U
0x0056	√	0x0056	LATIN CAPITAL LETTER V
0x0057	√	0x0057	LATIN CAPITAL LETTER W
0x0058	√	0x0058	LATIN CAPITAL LETTER X
0x0059	√	0x0059	LATIN CAPITAL LETTER Y
0x005A	√	0x005A	LATIN CAPITAL LETTER Z
0x005B	√	0x005B	LEFT SQUARE BRACKET
0x005C	√	0x005C	REVERSE SOLIDUS
0x005D	√	0x005D	RIGHT SQUARE BRACKET
0x005E	√	0x005E	CIRCUMFLEX ACCENT
0x005F	√	0x005F	LOW LINE

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x0060	√	0x0060	GRAVE ACCENT
0x0061	√	0x0061	LATIN SMALL LETTER A
0x0062	√	0x0062	LATIN SMALL LETTER B
0x0063	√	0x0063	LATIN SMALL LETTER C
0x0064	√	0x0064	LATIN SMALL LETTER D
0x0065	√	0x0065	LATIN SMALL LETTER E
0x0066	√	0x0066	LATIN SMALL LETTER F
0x0067	√	0x0067	LATIN SMALL LETTER G
0x0068	√	0x0068	LATIN SMALL LETTER H
0x0069	√	0x0069	LATIN SMALL LETTER I
0x006A	√	0x006A	LATIN SMALL LETTER J
0x006B	√	0x006B	LATIN SMALL LETTER K
0x006C	√	0x006C	LATIN SMALL LETTER L
0x006D	√	0x006D	LATIN SMALL LETTER M
0x006E	√	0x006E	LATIN SMALL LETTER N
0x006F	√	0x006F	LATIN SMALL LETTER O
0x0070	√	0x0070	LATIN SMALL LETTER P
0x0071	√	0x0071	LATIN SMALL LETTER Q
0x0072	√	0x0072	LATIN SMALL LETTER R
0x0073	√	0x0073	LATIN SMALL LETTER S
0x0074	√	0x0074	LATIN SMALL LETTER T
0x0075	√	0x0075	LATIN SMALL LETTER U
0x0076	√	0x0076	LATIN SMALL LETTER V
0x0077	√	0x0077	LATIN SMALL LETTER W
0x0078	√	0x0078	LATIN SMALL LETTER X
0x0079	√	0x0079	LATIN SMALL LETTER Y
0x007A	√	0x007A	LATIN SMALL LETTER Z
0x007B	√	0x007B	LEFT CURLY BRACKET
0x007C	√	0x007C	VERTICAL LINE
0x007D	√	0x007D	RIGHT CURLY BRACKET
0x007E	√	0x007E	TILDE
0x0082	Extension	0x201A	SINGLE LOW-9 QUOTATION MARK
0x0083	Extension	0x0192	LATIN SMALL LETTER F WITH HOOK
0x0084	Extension	0x201E	DOUBLE LOW-9 QUOTATION MARK
0x0085	Extension	0x2026	HORIZONTAL ELLIPSIS
0x0086	Extension	0x2020	DAGGER

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x0087	Extension	0x2021	DOUBLE DAGGER
0x0088	Extension	0x02C6	MODIFIER LETTER CIRCUMFLEX ACCENT
0x0089	Extension	0x2030	PER MILLE SIGN
0x008A	Extension	0x0160	LATIN CAPITAL LETTER S WITH CARON
0x008B	Extension	0x2039	SINGLE LEFT-POINTING ANGLE QUOTATION MARK
0x008C	Extension	0x0152	LATIN CAPITAL LIGATURE OE
0x0091	Extension	0x2018	LEFT SINGLE QUOTATION MARK
0x0092	Extension	0x2019	RIGHT SINGLE QUOTATION MARK
0x0093	Extension	0x201C	LEFT DOUBLE QUOTATION MARK
0x0094	Extension	0x201D	RIGHT DOUBLE QUOTATION MARK
0x0095	Extension	0x2022	BULLET
0x0096	Extension	0x2013	EN DASH
0x0097	Extension	0x2014	EM DASH
0x0098	Extension	0x02DC	SMALL TILDE
0x0099	Extension	0x2122	TRADE MARK SIGN
0x009A	Extension	0x0161	LATIN SMALL LETTER S WITH CARON
0x009B	Extension	0x203A	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK
0x009C	Extension	0x0153	LATIN SMALL LIGATURE OE
0x009F	Extension	0x0178	LATIN CAPITAL LETTER Y WITH DIAERESIS
0x00A0	√	0x00A0	NO-BREAK SPACE
0x00A1	√	0x00A1	INVERTED EXCLAMATION MARK
0x00A2	√	0x00A2	CENT SIGN
0x00A3	√	0x00A3	POUND SIGN
0x00A4	√	0x00A4	CURRENCY SIGN
0x00A5	√	0x00A5	YEN SIGN
0x00A6	√	0x00A6	BROKEN BAR
0x00A7	√	0x00A7	SECTION SIGN
0x00A8	√	0x00A8	DIAERESIS
0x00A9	√	0x00A9	COPYRIGHT SIGN
0x00AA	√	0x00AA	FEMININE ORDINAL INDICATOR
0x00AB	√	0x00AB	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
0x00AC	√	0x00AC	NOT SIGN
0x00AD	√	0x00AD	SOFT HYPHEN
0x00AE	√	0x00AE	REGISTERED SIGN
0x00AF	√	0x00AF	MACRON

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x00B0	√	0x00B0	DEGREE SIGN
0x00B1	√	0x00B1	PLUS-MINUS SIGN
0x00B2	√	0x00B2	SUPERSCRIPT TWO
0x00B3	√	0x00B3	SUPERSCRIPT THREE
0x00B4	√	0x00B4	ACUTE ACCENT
0x00B5	√	0x00B5	MICRO SIGN
0x00B6	√	0x00B6	PILCROW SIGN
0x00B7	√	0x00B7	MIDDLE DOT
0x00B8	√	0x00B8	CEDILLA
0x00B9	√	0x00B9	SUPERSCRIPT ONE
0x00BA	√	0x00BA	MASCULINE ORDINAL INDICATOR
0x00BB	√	0x00BB	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
0x00BC	√	0x00BC	VULGAR FRACTION ONE QUARTER
0x00BD	√	0x00BD	VULGAR FRACTION ONE HALF
0x00BE	√	0x00BE	VULGAR FRACTION THREE QUARTERS
0x00BF	√	0x00BF	INVERTED QUESTION MARK
0x00C0	√	0x00C0	LATIN CAPITAL LETTER A WITH GRAVE
0x00C1	√	0x00C1	LATIN CAPITAL LETTER A WITH ACUTE
0x00C2	√	0x00C2	LATIN CAPITAL LETTER A WITH CIRCUMFLEX
0x00C3	√	0x00C3	LATIN CAPITAL LETTER A WITH TILDE
0x00C4	√	0x00C4	LATIN CAPITAL LETTER A WITH DIAERESIS
0x00C5	√	0x00C5	LATIN CAPITAL LETTER A WITH RING ABOVE
0x00C6	√	0x00C6	LATIN CAPITAL LETTER AE
0x00C7	√	0x00C7	LATIN CAPITAL LETTER C WITH CEDILLA
0x00C8	√	0x00C8	LATIN CAPITAL LETTER E WITH GRAVE
0x00C9	√	0x00C9	LATIN CAPITAL LETTER E WITH ACUTE
0x00CA	√	0x00CA	LATIN CAPITAL LETTER E WITH CIRCUMFLEX
0x00CB	√	0x00CB	LATIN CAPITAL LETTER E WITH DIAERESIS
0x00CC	√	0x00CC	LATIN CAPITAL LETTER I WITH GRAVE
0x00CD	√	0x00CD	LATIN CAPITAL LETTER I WITH ACUTE
0x00CE	√	0x00CE	LATIN CAPITAL LETTER I WITH CIRCUMFLEX
0x00CF	√	0x00CF	LATIN CAPITAL LETTER I WITH DIAERESIS
0x00D0	√	0x00D0	LATIN CAPITAL LETTER ETH
0x00D1	√	0x00D1	LATIN CAPITAL LETTER N WITH TILDE

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x00D2	√	0x00D2	LATIN CAPITAL LETTER O WITH GRAVE
0x00D3	√	0x00D3	LATIN CAPITAL LETTER O WITH ACUTE
0x00D4	√	0x00D4	LATIN CAPITAL LETTER O WITH CIRCUMFLEX
0x00D5	√	0x00D5	LATIN CAPITAL LETTER O WITH TILDE
0x00D6	√	0x00D6	LATIN CAPITAL LETTER O WITH DIAERESIS
0x00D7	√	0x00D7	MULTIPLICATION SIGN
0x00D8	√	0x00D8	LATIN CAPITAL LETTER O WITH STROKE
0x00D9	√	0x00D9	LATIN CAPITAL LETTER U WITH GRAVE
0x00DA	√	0x00DA	LATIN CAPITAL LETTER U WITH ACUTE
0x00DB	√	0x00DB	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
0x00DC	√	0x00DC	LATIN CAPITAL LETTER U WITH DIAERESIS
0x00DD	√	0x00DD	LATIN CAPITAL LETTER Y WITH ACUTE
0x00DE	√	0x00DE	LATIN CAPITAL LETTER THORN
0x00DF	√	0x00DF	LATIN SMALL LETTER SHARP S
0x00E0	√	0x00E0	LATIN SMALL LETTER A WITH GRAVE
0x00E1	√	0x00E1	LATIN SMALL LETTER A WITH ACUTE
0x00E2	√	0x00E2	LATIN SMALL LETTER A WITH CIRCUMFLEX
0x00E3	√	0x00E3	LATIN SMALL LETTER A WITH TILDE
0x00E4	√	0x00E4	LATIN SMALL LETTER A WITH DIAERESIS
0x00E5	√	0x00E5	LATIN SMALL LETTER A WITH RING ABOVE
0x00E6	√	0x00E6	LATIN SMALL LETTER AE
0x00E7	√	0x00E7	LATIN SMALL LETTER C WITH CEDILLA
0x00E8	√	0x00E8	LATIN SMALL LETTER E WITH GRAVE
0x00E9	√	0x00E9	LATIN SMALL LETTER E WITH ACUTE
0x00EA	√	0x00EA	LATIN SMALL LETTER E WITH CIRCUMFLEX
0x00EB	√	0x00EB	LATIN SMALL LETTER E WITH DIAERESIS
0x00EC	√	0x00EC	LATIN SMALL LETTER I WITH GRAVE
0x00ED	√	0x00ED	LATIN SMALL LETTER I WITH ACUTE
0x00EE	√	0x00EE	LATIN SMALL LETTER I WITH CIRCUMFLEX
0x00EF	√	0x00EF	LATIN SMALL LETTER I WITH DIAERESIS
0x00F0	√	0x00F0	LATIN SMALL LETTER ETH
0x00F1	√	0x00F1	LATIN SMALL LETTER N WITH TILDE
0x00F2	√	0x00F2	LATIN SMALL LETTER O WITH GRAVE
0x00F3	√	0x00F3	LATIN SMALL LETTER O WITH ACUTE
0x00F4	√	0x00F4	LATIN SMALL LETTER O WITH CIRCUMFLEX

Table 8-40 – AS protocol code page

Codepoint	ISO/IEC 8859-1	Unicode code	Unicode name
0x00F5	√	0x00F5	LATIN SMALL LETTER O WITH TILDE
0x00F6	√	0x00F6	LATIN SMALL LETTER O WITH DIAERESIS
0x00F7	√	0x00F7	DIVISION SIGN
0x00F8	√	0x00F8	LATIN SMALL LETTER O WITH STROKE
0x00F9	√	0x00F9	LATIN SMALL LETTER U WITH GRAVE
0x00FA	√	0x00FA	LATIN SMALL LETTER U WITH ACUTE
0x00FB	√	0x00FB	LATIN SMALL LETTER U WITH CIRCUMFLEX
0x00FC	√	0x00FC	LATIN SMALL LETTER U WITH DIAERESIS
0x00FD	√	0x00FD	LATIN SMALL LETTER Y WITH ACUTE
0x00FE	√	0x00FE	LATIN SMALL LETTER THORN
0x00FF	√	0x00FF	LATIN SMALL LETTER Y WITH DIAERESIS

8.8.2 Font matching

An ASCE shall use the font matching algorithm described in this clause to determine the current set of matched fonts for all active ASCEs (including itself).

An ASCE shall determine the current set of matched fonts by comparing its local font information (i.e., the font attributes sent in its last FontPDU) against the font attributes provided by each other active ASCE, using the font match criteria described in Table 8-41 below. The match criteria are successively applied to the FontPDU parameters in a series of pairs of fonts and a font is added to the current set of matched fonts where a match is found at all other active ASCEs. This means that:

- all active ASCEs have to match a font for it to be added to the current set of matched fonts;
- where one other active ASCE has not yet provided font attributes, there are no current matched fonts.

The output of font matching is a mapping for each matched font between local and remote FontIDs for each other active ASCE. See clause 8.8 for further information on FontIDs. The matched font mappings are used as follows:

- When sending text and extended text orders for a matched local font, the sending ASCE shall set the order FontID parameter to correspond to that font's FontID in the last sent FontPDU.
- When receiving text and extended text orders, the receiving ASCE shall map the FontID in the order to the local font that matched that FontID in the last received FontPDU from that ASCE.

This mapping mechanism allows ASCEs to use a single set of FontIDs when sending text and extended text orders and places the responsibility on receiving ASCEs to map from FontID in the order to the local font on that terminal.

Use of the same font matching algorithm ensures that each pair of ASCEs agree on an identical set of matched fonts and therefore generate an identical set of interoperable font mappings.

Table 8-41 describes the minimum match criteria for each font parameter (corresponding to the parameter in the relevant FontPDU) considered during font matching, described in terms of the local font and one other remote font.

Table 8-42 provides information on the allowable match types referenced in Table 8-41. An ASCE may or may not allow approximate matches, based on a purely local mechanism (such as local terminal configuration options). Where an ASCE does not allow approximate matches, only exact matches are permitted. Where an ASCE does allow approximate matches, it may do so on a per-match criteria basis (again dependent on local mechanisms), provided the corresponding negotiated capabilities allow that match type.

A pair of fonts are defined to match where each of the font parameters requiring consideration in Table 8-41 are either all exact matches or meet the minimum match types defined in Table 8-41 (dependent on the ASCE's local approximate match configuration). For example, a particular ASCE may allow Delta X position but not code page approximate matches and, provided the negotiated capabilities allow for Delta X simulation, may then require that a pair of fonts are exact matches with respect to those considered parameters defined as requiring exact and code page minimum match types, but only require an approximate match for those considered parameters defined as requiring Delta X position minimum match types.

Table 8-41 – Font match criteria

Parameter	Criteria	Match type
faceName	The current and remote face names are identical.	Exact
fontFlags fixed pitch	Both fonts are fixed pitch OR both fonts are proportional (not fixed pitch).	Exact
codePage	Both fonts support all codepoints. Either the local or remote font only supports core codepoints.	Exact Code page
fontFlags fixed size	Both fonts are fixed size OR both fonts are scalable (not fixed size). The local font is fixed size and the remote font is scalable. (Note 1)	Exact Exact
averageWidth and height	Both fonts are fixed size and the averageWidth and height values are the same. (Note 2)	Exact
signature1, signature2 and signature3	These criteria depend on the negotiated capabilities for font signature checking (i.e., in legacy mode on the negotiated Order.textFlags capability check font signatures bit flag; in base mode on the negotiated Order.checkFontSignatures capability). • Font signature checking is enabled and the local and remote signature1, signature2 and signature3 values are the same. • Font signature checking is enabled and the local and remote signature1 and signature2 values are the same. • Font signature checking is enabled and the local and remote signature1, signature2 and signature3 values are not the same or either font has the special value NO_SIGNATURE. • Font signature checking is not enabled.	 Exact Delta X position Delta X position Delta X position

Table 8-41 – Font match criteria

Parameter	Criteria	Match type
aspectX and aspectY	This criteria depend on the negotiated capabilities for font aspect checking (i.e., in legacy mode on the negotiated Order.textFlags capability check font aspect bit flag; in base mode on the negotiated Order.checkFontAspectFlag capability).	
	• Font aspect checking is enabled and the local and remote aspectX and aspectY values are the same.	Exact
	• Font aspect checking is not enabled.	Delta X position
NOTE 1 – This assumes that an ASCE may match local fixed size fonts with remote scalable fonts and subsequently send text and extended text orders for that font, relying on receiving ASCEs to scale the text appropriately. It assumes that the inverse (i.e., matching scalable to fixed size and sending scalable) is not allowed. NOTE 2 – If either font is scalable, these criteria are not considered.		

Table 8-42 – Font match types

Match type	Definition
Exact	Both font parameters under consideration must match exactly.
Code page	Where one or both of the code pages for the fonts under consideration only conforms to the core AS protocol code page, an ASCE may allow an approximate code page match. Where it does so, it shall subsequently restrict codepoints for that font in text and extended text orders to the core AS protocol code page. See clause 8.16.11 and clause 8.16.12 for further information on text and extended text orders.
Delta X position	Where positioning information for one or both of the fonts under consideration does not match exactly, an ASCE may allow an approximate Delta X position match. An ASCE shall only allow approximate Delta X matches where the negotiated capabilities allow for Delta X simulation (i.e., in legacy mode, the Order.textFlags allow DeltaX bit flag is set; in base mode, the Order.allowDeltaXFlag is TRUE) and shall thereafter simulate text X positioning information for the font using Delta X information in extended text orders. See clause 8.16.12 for further information on the extended text order and Delta X simulation.

8.8.3 Font aliasing

Font matching acts on font attributes in FontPDUs. This enables an ASCE to perform font aliasing between fonts on the local terminal and the font attributes it sends in FontPDUs. This is a local ASCE implementation decision, but is particularly useful in the following scenarios:

- The same (or very similar) fonts are available from a range of font suppliers for the local terminal (but use different face names). Here, an ASCE may map the local terminal fonts to a generic (supplier-independent) face name to increase the probability of font matching.
- The same (or very similar) fonts have different face names on different terminal types. Here an ASCE may map the local terminal fonts to a new (or additional) face name to increase the probability of font matching.

For example, consider two ASCEs, where ASCE A supports Font_Supplier_A:Courier and Font_Supplier_B:Courier and ASCE B supports just Courier. If ASCE A provides font attributes for Font_Supplier_A:Courier, Font_Supplier_B:Courier and two aliased Couriers (corresponding to the two supplier-specific Couriers), then (provided other criteria match) both ASCEs are able to send orders for some or all local Courier text.

8.9 Application management

During an AS session, an ASCE may host one or more applications, each of which consists of a collection of hosted windows that are shared to peer active ASCEs.

An ASCE shall notify peer active ASCEs when the number of hosted applications on the ASCE's local terminal changes, by sending an ApplicationPDU with action NotifyHostedApplications containing the new number of hosted applications to all ASCEs in the conference, in the manner indicated in Table 6-3. The content of the ApplicationPDU is shown in Table 8-43.

This provides ongoing information to other active ASCEs on the number of applications hosted by this ASCE, which they may use to provide end-user information. Receiving ASCEs may use this ASPDU to monitor when remote ASCEs stop hosting applications – which may allow them to free up resources allocated for each other hosting ASCE. See clause 8.6 for further information on synchronization.

An ASCE may start and stop hosting local applications at any time. This is purely a local matter and may be driven by the local end-user, local terminal behaviour and/or by conference initialization/termination.

However, there are situations where it is useful to unhost remote applications. For example, where an ASCE is controlling hosted applications that were remote shared (see clause 8.7) or when the remote end-user is inexperienced. An ASCE may require a peer ASCE to unhost an application hosted on the peer ASCE's terminal by sending an ApplicationPDU with action UnhostApplication containing a windowID for the application to be unhosted in the manner indicated in Table 6-3. The supplied windowID should be a hosted_window_ID belonging to the application obtained from the most recent WindowListPDU sent by the hosting ASCE (see clause 8.10). An ASCE shall only send an ApplicationPDU with action UnhostApplication where the peer ASCE's General.remoteUnshareFlag capability is TRUE.

On receipt of an ApplicationPDU with action UnhostApplication containing a windowID for an application hosted on the local terminal, an ASCE shall cease hosting the application concerned.

Table 8-43 – ApplicationPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
action	This parameter identifies the particular ApplicationPDU action. Allowable values are NotifyHostedApplications or UnhostApplication.
numberApplications	This parameter indicates the number of applications hosted by the sending ASCE. This parameter is only valid where the action is NotifyHostedApplications.
windowID	This parameter specifies the top-most window owned by an application hosted on the peer ASCE to be unhosted. This parameter is only valid where the action is UnhostApplication.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.10 Window list management

During an AS session, where an ASCE is hosting one or more windows that are shared to peer active ASCEs, it is managing a collection of windows, where those windows may be:

- Hosted: Hosted windows are owned by an application hosted on the local terminal. For each hosted window, there is a corresponding shadow window on each peer ASCE.
- Shadow: Shadow windows are drawn by the ASCE and correspond to a hosted window on a particular peer ASCE.
- Local: Local windows are not shared – their application output is only visible on the local terminal.

An ASCE is only required to track visible hosted windows. On certain terminals and/or terminal window managers, end-user or programmatic action may make application windows temporarily invisible (i.e., it is removed from the local terminal display) – for example, where a long-running system monitor application hides itself until an alarm event occurs that requires user action, whereupon it becomes visible again. Where a local hosted window becomes invisible, then an ASCE is not required to track the window and should not provide window list update information for the window (see below).

Where a hosted window is obscured by local window(s) and the ASCE cannot obtain valid drawing information for that hosted window, the AS protocol requires that the hosting ASCE includes the obscuring local window(s) in the appropriate window list (see below) and that receiving ASCEs mark such obscured areas (in a locally determined manner) to indicate that the contents are not necessarily valid.

Note that an ASCE need only track local windows when hosting and need only track those that obscure hosted windows and prevent it obtaining valid drawing information from those hosted windows. For example, an ASCE may display hosted and shadow windows in a dedicated terminal display area from which local windows are always excluded. Similarly, the particular terminal may not support local applications (and therefore windows) at all. Or, on certain terminal equipment, an ASCE may still be able to obtain valid drawing information from hosted windows where they are obscured by local windows. For these cases, an ASCE is not required to track local windows.

An ASCE shall send a WindowListPDU to all ASCEs in the conference in the manner indicated in Table 6-3, when:

- it starts or stops hosting windows;
- it is hosting windows and detects a change in the visible hosted window Z-order;
- it is hosting windows and one or more of its visible hosted window positions change;
- it is hosting windows and detects a local window Z-order or position change affecting the obscuring of visible hosted windows.

This means that an ASCE only sends a WindowListPDU when it is hosting visible windows, or, as in the first case, when it is making the transition to or from hosting windows. The content of the WindowListPDU is shown in Table 8-44. The WindowListPDU contains information on two classes of windows:

- The sending ASCE's view of all visible hosted or shadow windows within the conference. This view is (generally – but see below) common to all active ASCEs, since it includes hosted and shadow windows – and a window included in its list by one ASCE because it is hosted will be included in the corresponding lists by other ASCEs because it is shadowed.
- Any of the sending ASCE's local windows that obscure at least part of at least one visible hosted window. This information is owned and generated by a particular ASCE – since it relates to local windows, there is no overlap with similar information generated by peer ASCEs.

The WindowListPDU contains a single window list, containing hosted and shadow windows and those local windows that meet the obscuring criteria, ordered such that the first window corresponds to the top (front-most) window and the last corresponds to the bottom (back-most) window in the local terminal Z-order.

For each window in the window list, the ASCE provides Z-order information (implicitly by position in the list), position and size, ownership information and any appropriate qualifiers (such as whether the window is minimized).

The AS protocol assumes that the information provided for hosted and shadow windows in the window list should normally be identical on all active ASCEs. It does not support independent Z-order, size/positioning or qualification (such as minimized) of shadow windows with respect to their corresponding hosted window. That is, where a hosted window's Z-order, size/position or qualification changes, then the AS protocol requires that an equivalent change is applied to corresponding shadow windows on all other active ASCEs.

In practice, transient window list differences in hosted and shadow windows may occur, where the WindowListPDU reflecting a window list change has not yet been received and/or processed by other ASCEs. Much of the complexity in AS window list handling relates to the resolution of races in window list updates. The following description ignores the possibility of window list races and their resolution, which is covered in clause 8.10.1.

On receipt of a WindowListPDU, an ASCE typically performs the following operations – although the specifics will depend on the local terminal environment.

- It processes the WindowListPDU window list to:
  - remove any old shadow windows;
  - create any new shadow windows;
  - update the size and/or position of windows that have been changed;
  - adjust each hosted or shadow window's position in the Z-order on the local terminal to give the same relative Z-order as in the window list, taking care to maintain the correct Z-order relationship to local applications.
- It determines the new obscured window region – which determines which areas of shadow windows are valid for drawing – based on the sending ASCE's desktop size and any obscuring local windows in the WindowListPDU.

An ASCE is required to apply window list changes as accurately as possible within the constraints of the local terminal environment.

Table 8-44 – WindowListPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
listTime	This parameter is the local ASCE time in milliseconds when this WindowListPDU was constructed. It is used in association with the listID parameter (see below) to resolve WindowListPDU races. See clause 8.10.1 below for further information.
listID	This parameter is the identifier assigned by the sending ASCE when this WindowListPDU was constructed. It is used in association with the listTime parameter (see above) to resolve WindowListPDU races. See clause 8.10.1 below for further information.

Table 8-44 – WindowListPDU

Parameter	Description
windowAttributeList	This parameter is a list of window attributes (see Table 8-45) describing the window structure on the sending ASCE. The list is in Z-order, such that the first window is the top (front-most) and the last is the bottom (back-most) in the local terminal Z-order.
windowTitleList	This parameter is a list of window titles. The list is in Z-order, such that the first window is the top (front-most) and the last is the bottom (back-most) in the local terminal Z-order. Each title is either the special value NO_TITLE or is a T.50 text string describing the window.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

Many terminal window managers provide local facilities (such as a task list, task bar or icon tray) to allow end-users to activate or arrange windows. Such facilities are typically restricted to top-level windows – so that end-users can view the collection of windows that comprise the application as a whole. An ASCE may provide a title for a hosted window where the title would help end-users at peer ASCEs more readily identify the corresponding shadow window. It is recommended that an ASCE should only supply titles for hosted windows. Where the window is shadow or local, the ASCE should supply the special NO_TITLE title string.

The following requires that an ASCE assigns a unique local identifier for each hosted application (its hosted_application_ID) and a unique local identifier for each hosted window (its hosted_window_ID).

Table 8-45 – Window attributes

Parameter	Description
windowID	This parameter is the window identifier for this window. • Where the window is hosted, this parameter is the hosted_window_ID assigned by the sending ASCE. • Where the window is a shadow window, this parameter is the hosted_window_ID assigned by the owning ASCE (which was supplied in the last WindowListPDU for the corresponding hosted window).
windowExtra	This parameter supplies additional information for this window. • Where the window is hosted, this parameter is the hosted_application_ID assigned by the sending ASCE. • Where the window is a shadow window, this parameter is the MCS user ID of the hosting ASCE.
windowOwner	This parameter identifies the owning window for this window. • Where the window is hosted, this parameter is the hosted_window_ID of the owning window (i.e., the window that is the parent of this window in the local terminal window hierarchy). Note that where the window is hosted and the owning window is the desktop, this parameter shall be zero.

Table 8-45 – Window attributes

Parameter	Description
windowFlags	This parameter is a set of bit flags qualifying the window. Defined bit flag values are as follows: • Minimized (hosted windows only). • Taggable (hosted windows only). • Hosted. • Shadow. • Local. • Always on top (hosted windows only). • Window manager minimized (hosted windows only). • Window manager invisible (hosted windows only). See below for further information on the interpretation of window flags.
windowLeft	This parameter is the left X coordinate of the window on the virtual desktop. For hosted and local windows, this parameter shall be the left X coordinate of the window (which may be outside the local desktop).
windowTop	This parameter is the top Y coordinate of the window on the virtual desktop. For hosted and local windows, this parameter shall be the top Y coordinate of the window (which may be outside the local desktop).
windowRight	This parameter is the right X coordinate of the window on the virtual desktop. For hosted and local windows, this parameter shall be the right X coordinate of the window (which may be outside the local desktop).
windowBottom	This parameter is the bottom Y coordinate of the window on the virtual desktop. For hosted and local windows, this parameter shall be the bottom Y coordinate of the window (which may be outside the local desktop).
nonStandardWindowAttributes	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard window attributes allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

Taggable

End-users may have difficulty identifying which windows belong to which user in the conference. This is an issue where the end-user selects an application feature that will take effect on the hosting terminal (e.g., saving a file) and/or where multiple ASCEs within a conference are hosting applications, increasing the number of windows on the local terminal desktop. The window attribute taggable bit flag provides a mechanism whereby ASCEs may provide additional user interface information about hosted window location.

An ASCE constructing a window list update should use the taggable bit flag to indicate whether hosted window(s) are eligible for tagging. It is recommended that sending ASCEs should only flag top-level windows as taggable and should not tag subsidiary and transient windows such as help hints or icon titles. The algorithm used to determine whether a window is taggable is purely a local matter, but it is recommended that ASCEs should mark as taggable only those windows that a user

would perceive as constituting a distinct main application window or dialogue. An ASCE receiving a window list containing hosted windows with the taggable bit flag set may provide additional per-shadow window user interface cues (such as window tags and/or augmentation of the window title) identifying the node owning the corresponding hosted windows.

Minimized windows

Certain terminals support the concept of minimization of windows, whereby a window is reduced to an icon or smaller window, which may then be displayed on the local terminal desktop or in a window owned by the terminal window manager (such as a window manager task list, task bar or icon tray). Where an application consists of multiple windows, then minimizing the application typically results in the application's top-level window(s) being minimized and other non-top-level windows being made invisible.

Various styles of window minimization for hosted windows may be represented in window list updates using several of the windows attributes windowFlags – namely the minimized, window manager minimized and window manager invisible bit flags.

- The minimized bit flag should be set by an ASCE when a hosted window is minimized – irrespective of the particular local minimization behaviour.
- The window manager minimized bit flag should be set by an ASCE when a hosted window is minimized to a local window manager window. This bit flag is typically applied to top-level windows.
- The window manager invisible bit flag should be set by an ASCE when a hosted window is made invisible as part of a local window manager minimization mechanism and where the window manager minimized bit flag is set for a parent window – i.e., where the main top-level hosted windows of an application are minimized to a local window manager window. It should not be set where the top-level hosted windows are minimized to the desktop.

The use of these bit flags does not presume a particular local terminal window minimization behaviour or a particular local terminal window manager. Hosting ASCEs should set the allowable combination of bit flags that best expresses the characteristics of each minimized hosted window. Similarly, an ASCE processing a window list update containing hosted windows with one or more of these bit flags set should present the corresponding shadow windows in a manner that is appropriate to the local terminal and/or window manager minimization behaviour. This approach allows ASCEs to support the various minimization styles across multiple terminal types, with each ASCE presenting minimized shadow windows in a manner that is appropriate for the local terminal.

Always on top

Certain terminal types support the concept of always on top windows, where such windows are normally displayed on top of all other windows. For example, always on top windows are often used for application suite tool bars. Where the local terminal supports always on top windows, there may be multiple windows which are treated as always as top, which are typically not part of the main terminal Z-order, but are rather managed as members of a separate always on top Z-order. Within this separate Z-order, always on top windows may change z-order with respect to each other, but are always on top of ordinary (i.e., not always on top) windows.

Where always on top windows are supported on a particular terminal, the always on top bit flag should be set by an ASCE when a hosted window is treated locally as being always on top. This bit flag is typically applied to top-level hosted windows. ASCEs should also ensure that hosted windows marked as being always on top are higher than (i.e., in front of) windows that are not marked as such in the WindowListPDU list of windows. An ASCE processing a window list update containing hosted windows with the always on top bit flag set should attempt to make the corresponding shadow windows behave as always on top in the local terminal environment – which

may be achieved using a similar facility in the local terminal environment or by manipulating the local Z-order to achieve the desired effect.

8.10.1 Window list Z-order races

As discussed above, each active ASCE within a conference maintains a window list for hosted, shadow and obscuring local windows. As multiple ASCEs may report window list updates to other ASCEs via the WindowListPDU, there is always the possibility of a race. Creation and deletion of windows and modifications to their size and position can only occur on the terminal hosting application owning the window concerned. In contrast, the position of a shared window in the Z-order can be altered on any terminal.

If WindowListPDU window lists were simply applied by recipient ASCEs, this would result in convergence for window existence, position and size (since the update in the two colliding window lists would be orthogonal). However, naive application of Z-order changes can result in indefinite cross-over of window lists. Therefore, each ASCE maintains a window list identifier which it places in each WindowListPDU for Z-order race resolution. The identifier has three parts:

- Sequence number: This is the current value of the sequence number as calculated by the sending ASCE (see below); this is held in bits 4-15 (i.e., the most significant three nibbles) of the listID parameter.
- Increment: This is the absolute amount by which the ASCE incremented the last seen sequence number to generate the new sequence number (see below); this is held in bits 0-3 (i.e., the least significant nibble) of the listID parameter.
- Tick: This is a tick in milliseconds, which is used to resolve races when the sequence number is identical; this is held in the listTime parameter.

Each ASCE calculates the next identifier value by incrementing the last seen (i.e., sent or received) sequence number by the priority of the window list change to create a new sequence number. The new identifier is formed from the new sequence number, the increment (or priority) and the current local time in milliseconds. The recommended increment/priority values are as follows:

- 0 $\Rightarrow$ No change in Z-order but changes in window positions and/or sizes.
- 2 $\Rightarrow$ Change in Z-order.
- 3 $\Rightarrow$ Change in Z-order and change in window activation.
- 4 $\Rightarrow$ Change in Z-order and change in window activation to special window (see clause 8.10.2).
- 5 $\Rightarrow$ ASCE is no longer hosting.

An increment of 5 should only be used when an ASCE stops hosting applications to force a timely removal of all corresponding shadow windows.

Window list updates with a zero increment identifier are always applied as they do not affect the Z-order – and are therefore not subject to race resolution. However, window list identifiers with increments 2 through 5 require that the receiving ASCE check for Z-order races, which are resolved as follows:

- The application of a received window list may cause changes already present in the local system (but not yet sent) to be lost. To minimize temporary inconsistencies and to minimize "rebounds", an ASCE should simulate such collisions by testing the received identifier against the (forecast) identifier that it would use in its next WindowListPDU. If there are pending window changes and the forecast identifier is later than the last identifier sent/received, then the received window list Z-order is not applied and the next window list (containing the pending changes) is duly sent.

- If there are pending window changes and the forecast identifier is earlier than the last identifier sent/received, then the received window list Z-order is applied in the normal way, and the received identifier becomes the new identifier. Therefore, when the next window list is generated, it will be sent with an identifier that does not indicate a collision.
- If there are no pending window changes, the receiving ASCE compares the received identifier against the last identifier seen. There are two cases to consider:
  - If the received identifier is later, the received window list Z-order is applied and the ASCE should schedule the conditional sending of a WindowListPDU to resend pending window changes.
  - If the received identifier is earlier, then the received window list Z-order is discarded.

Identifiers are compared using the following rule. Identifier A is later than Identifier B if:

- A's sequence number > B's sequence number.
- A's sequence number = B's sequence number AND A's increment > B's increment.
- A's sequence number = B's sequence number AND A's increment = B's increment AND A's tick > B's tick.

The final test for the final case (A's tick > B's tick) is used solely for race resolution – it does not imply any particular global time ordering.

8.10.2 Implementation considerations

On certain terminals, critical window manager error boxes (which appear to be windows) are actually drawn directly to the desktop window. An ASCE should handle this scenario by creating a virtual full screen shared window at the front of the Z-order. When peer ASCEs draw this virtual window, it will appear to be transparent for the areas that do not actually contain the error box, as updates will only be received for the actual error box area (most of the full screen window is not actually painted). When the error box is dismissed, the virtual window is deleted and everything returns to normal.

Certain terminals support multiple screen sessions, where one or more screen sessions may be full-screen text-based sessions which may not be accessible to the ASCE. Where such a full-screen text session has control of the screen, the ASCE should add a virtual local full screen window at the front of the window list in the next WindowListPDU. This will be interpreted by peer ASCEs as a window that totally obscures all of the hosted windows, with the consequence that all shadow windows on recipient ASCEs will be within the obscured area.

8.11 Window activation

Many terminals and/or terminal window managers support a concept of window activation, whereby a particular local window is considered to be the active window – this is often referred to as the focus window or the input focus window. Where such a concept is supported, then the active window typically receives all local terminal pointing device and keyboard input.

The AS protocol provides a window activation mechanism, whereby ASCEs may indicate changes in local window activation state and request changes in window activation state on other ASCEs. An ASCE is required to map the local terminal model of window activation onto the AS protocol model described in this clause. Where the local terminal does not support window activation, then the ASCE may perform a trivial mapping.

Certain terminals and/or terminal window managers allow an application to capture the pointing device. Where this is the case, the capturing application window may not be the active window (e.g., the capturing application window may minimize itself or become invisible while the capture is in progress), but continues to receive the captured pointing device input. In the AS protocol, pointing device capture by a hosted window is treated as a special case (see below).

AS window activation is tightly integrated with conductorship (if in effect within the conference) and with the conference AS control state. If an ASCE does not have the right to provide input to hosted or shadow windows, then it cannot change window activation state within the conference. See clause 8.19 for further information on conducted mode and clauses 8.12 and 8.13 for further information on control mechanisms.

When the active window changes on a terminal within the conference, the new active window may be any one of:

- a visible, invisible or capturing hosted window;
- a visible or invisible local window;
- a visible shadow window (ASCEs do not typically maintain invisible shadows).

8.11.1 Activation indications and requests

An ASCE shall monitor for two classes of activation events – referred to as indications and requests:

- Indications: These are activation or capture state changes to a local or hosted window. These affect a window owned by an application on the local terminal and need to be propagated to peer ASCEs. When an ASCE detects an activation or capture state change state affecting a local or hosted window, then it shall send a WindowActivationPDU with an appropriate indication (see below) to all ASCEs in the conference in the manner indicated in Table 6-3. The content of the WindowActivationPDU is shown in Table 8-46.
- Requests: These are activation changes to (or corresponding to) shadow windows. Some terminal window managers provide terminal-specific activation sequences, such as the ALT-TAB keyboard sequence cycling the window focus or window manager context menus that allow an end-user to switch focus to a specific window. Where the affected window is a shadow, the change should not be performed locally, but should be redirected to the ASCE that owns the corresponding hosted window. Where an ASCE recognizes such a change to a shadow window, it shall send a WindowActivationPDU with an appropriate request (see below) to the peer ASCE hosting the corresponding hosted window in the manner indicated in Table 6-3.

Changes in the window activation status of hosted windows may result in window list changes. For example, a restore window request will normally result in the corresponding hosted window changing its position and/or Z-order. In contrast, a hosted window active indication may or may not result in a Z-order change, dependent on the hosting ASCE's terminal window manager focus policy.

Changes in the window activation status of hosted windows do not cause changes in control.

Table 8-46 – WindowActivationPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
action	This parameter identifies the particular WindowActivationPDU action. The defined range of actions is classified as either indications or requests. The allowable actions are as follows. • Local window active (indication). • Hosted window active (indication). • Hosted window invisible (indication). • Pointing device capture (indication). • Activate window (request).

Table 8-46 – WindowActivationPDU

Parameter	Description
	• Close window (request). • Restore window (request). • WindowManagerMenu (request). • ActivationHelpKey (request). • ActivationHelpIndexKey (request). • ActivationHelpExtendedKey (request). • See below for further information.
activationID	This parameter is an identifier assigned by the sending ASCE. This parameter is only valid for indications where it is used in association with the defined indication priorities to resolve WindowActivationPDU races. The allowable range of values is 1..65534. See clause 8.11.2 below for further information.
activationWindow	This parameter provides an optional windowID for specific message types (see below).
activationPoint	For WindowManagerMenu requests, this is an optional parameter which may contain the position of the pointing device that caused the activation request.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

The following describes each WindowActivationPDU indication and request in further detail. Some of the required actions require that an ASCE be able to change the local terminal activation to no window having the local focus. On some terminal types, this may require that an ASCE maintain a special (invisible) no focus window. The following requires (as in clause 8.10) that an ASCE assigns a unique local identifier for each hosted window (its hosted_window_ID).

- The local window active indication indicates that activation has changed to a local window on the sending ASCE. On receipt of a local window active indication, an ASCE shall change the local activation to no focus.
- The hosted window active indication indicates that activation has changed to a hosted window on the sending ASCE. The WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window that is now active. On receipt of a hosted window active indication, an ASCE shall change the local activation to the local shadow window that corresponds to the indicated hosted window.
- The hosted window invisible indication indicates that activation has changed to an invisible hosted window on the sending ASCE. On receipt of a hosted window invisible indication, an ASCE shall change the local activation to no focus.
- The pointing device capture indication indicates that activation has changed to a hosted window that has captured the pointing device on the sending ASCE. Where the capturing hosted window is visible, the WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window that has captured the pointing device. Where the capturing hosted window is not visible, the WindowActivationPDU activationWindow parameter is zero. On receipt of a pointing device capture indication with a hosted_window_ID, an ASCE shall change the local activation to the local shadow window that corresponds to the indicated hosted window. On receipt of a pointing device

capture indication without a hosted_window_ID, an ASCE shall change the local activation to no focus.

- The activate window request indicates that the sending ASCE has detected a local request to activate a shadow window. The WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window on the peer ASCE that corresponds to the shadow window on the sending ASCE. On receipt of an activate window request, an ASCE shall activate the indicated hosted window. This should result in the receiving ASCE sending a subsequent WindowActivationPDU hosted window active indication for the hosted window.
- The close window request indicates that the sending ASCE has detected a local request to close a shadow window. The WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window on the peer ASCE that corresponds to the shadow window on the sending ASCE. On receipt of a close window request, an ASCE shall close the indicated hosted window. This should result in the receiving ASCE sending a subsequent WindowActivationPDU indication for the local window that inherits activation.
- The restore window request indicates that the sending ASCE has detected a local request to restore a shadow window. The WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window on the peer ASCE that corresponds to the shadow window on the sending ASCE. On receipt of a restore window request, an ASCE shall restore the indicated hosted window. This should result in the receiving ASCE sending a subsequent WindowActivationPDU hosted window active indication for the hosted window.
- The WindowManagerMenu request indicates that the sending ASCE has detected a local window manager menu-related operation corresponding to a minimized hosted window. The WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window on the peer ASCE that corresponds to the shadow window on the sending ASCE and the activationPoint parameter optionally contains the x and y virtual desktop coordinates of the activation event (where it was initiated by the pointing device). On receipt of a WindowManagerMenu request, an ASCE shall submit a locally equivalent menu request for the indicated hosted window.
- The ActivationHelpKey, ActivationHelpIndexKey and ActivationHelpExtendedKey requests indicate that the sending ASCE has detected the corresponding local window manager help activation request for a shadow window. The WindowActivationPDU activationWindow parameter contains the hosted_window_ID of the hosted window on the peer ASCE that corresponds to the shadow window on the sending ASCE. On receipt of an ActivationHelpKey, ActivationHelpIndexKey or ActivationHelpExtendedKey request, an ASCE shall submit the locally equivalent help activation request for the indicated hosted window.

An ASCE may only be able to detect certain activation requests on certain terminals and/or may only be able to process certain activation requests on certain terminals. For example, while terminal window manager activation is typically specific to the terminal and/or window manager type, the underlying concept is still sufficiently generic that an ASCE may detect the activation locally (in a manner specific to terminal window manager A) and a peer ASCE may process it (in a manner specific to terminal window manager B). In contrast, help activation keys (which switch to a specific class of help window) are less widely implemented and may only be available when both ASCEs (i.e., the ASCE with the shadow window and the peer ASCE with the corresponding hosted window) are executing on the same terminal type:

- An ASCE shall only send a WindowActivationPDU with a WindowManagerMenu action, where the peer ASCE's window Activation.windowManagerMenuFlag capability is TRUE.

- An ASCE shall only send a WindowActivationPDU with an ActivationHelpKey action, where the peer ASCE's window Activation.HelpKeyFlag capability is TRUE.
- An ASCE shall only send a WindowActivationPDU with an ActivationHelpIndexKey action, where the peer ASCE's window Activation.HelpExtendedKeyFlag capability is TRUE.
- An ASCE shall only send a WindowActivationPDU with an ActivationHelpExtendedKey action, where the peer ASCE's window Activation.HelpExtendedKeyFlag capability is TRUE.

8.11.2 Activation identifiers and priorities

The asynchronous nature of the AS activation protocol and the multiplicity of reasons for window activation state changes means that WindowActivationPDU collisions can occur. To detect these collisions, and hence maintain a consistent activation state across all active ASCEs in a conference, an ASCE shall place an activation identifier in all WindowActivationPDU indications. Identifiers are not required for WindowActivationPDU requests.

Each ASCE in a conference maintains the last identifier sent or received in a WindowActivationPDU indication and increments the identifier (in the range 1..65534 with rollover) when sending each WindowActivationPDU indication:

- If a received indication has an identifier less than the last sent or received identifier, it is discarded.
- If a received indication has an identifier equal to the last sent or received identifier, then the received indication's priority (see below) is compared to the priority of the last indication sent or received. If the priority of the received indication is less than or equal to the priority of the last sent or received indication, then the received indication is discarded. Otherwise, it is applied.
- If a received indication has an identifier greater than the last sent or received identifier, then it is applied.

Table 8-47 describes the defined priorities for WindowActivationPDU indications.

Identifiers are not used for WindowActivationPDU requests. This ensures that requests are applied independently of the indication identifier and priority handling. For example, where an ASCE sends an activate window request closely followed by a restore window request, then the second request is still applied, even though it may postdate a hosted window active indication from the receiving ASCE.

Table 8-47 – WindowActivationPDU indication priorities

Indication	Priority
Local window active	1
Hosted window active	1
Hosted window invisible	1
Pointing device capture	2

8.12 Control

The conference control policy is a major determinant of application sharing usability. Experience shows that different control policies are applicable for different mixes of conference size and/or end-user experience. Therefore, the AS protocol does not mandate a particular control policy, but rather provides a set of core control mechanisms whereby ASCEs can implement a range of policies with (potentially) different characteristics – either sequentially or concurrently within the

conference. The AS protocol also defines an additional mediated set of control mechanisms, which build upon the core control mechanisms described in this clause – see clause 8.13 for further information.

The core AS control protocol is based on managing the right to provide input to hosted and/or shadow windows. In combination, these rights support the following core control modes:

- Detached: In this mode, an ASCE:
  - has the right to provide input to hosted windows;
  - does not have the right to provide input to shadow windows;
  - denies peer ASCEs the right to provide input to shadow windows that correspond to hosted windows on this ASCE.

  In practice, this allows an end-user to work with hosted applications without interference from other users – other users cannot supply input, activation changes or Z-order changes.
- Cooperating: In this mode, cooperating ASCEs within the conference serially acquire the right to provide input to hosted and shadow windows. At any point in time within the conference:
  - one of the cooperating ASCEs can provide input to hosted and shadow windows – but only where other ASCEs are not detached (it is "in control");
  - the other cooperating ASCEs cannot provide input to hosted and shadow windows (they are "viewing").

Where an ASCE does not have the right to provide input to shadow windows – it is detached or cooperating/viewing – it may still provide information on pointing device movement to other ASCEs. Where an ASCE is in one of these control states and still provides this information, then other ASCEs may use the information to provide end-user feedback on the sending ASCE's pointing device activity – which may (for certain terminal types) substantially improve the remote end-users' perception of application sharing usability. Whether to provide the pointing device information when in these states, and how to present it on reception, is an ASCE implementation decision. See clause 8.18 for further information on input and pointing device events.

The core AS control protocol does not specify an ASCE's or local terminal's rights to provide input to local windows when in either detached or cooperating modes. The particular policy adopted for local input rights will normally be determined by the particular characteristics of the local terminal.

A conference may contain any mix of detached and cooperating ASCEs. ASCEs may freely move between cooperating and detached modes. In contrast, an ASCE can only be in control while it owns the control identifier. Each ASCE (whether cooperating or detached) tracks the current control identifier value within the conference and which ASCE currently holds the identifier.

When an ASCE wishes to change the control state of the conference, it shall send a ControlPDU to all ASCEs in the conference in the manner indicated in Table 6-3. The content of the ControlPDU is shown in Table 8-48.

Table 8-48 – ControlPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
action	This parameter identifies the particular ControlPDU action. The allowable actions are as follows. • Request control. • Grant control.

Table 8-48 – ControlPDU

Parameter	Description
	• Detach. • Cooperate. See below for further information.
grantID	When the action parameter (see above) is grant control, this parameter specifies the MCS user ID of the ASCE being granted control – i.e., the new control identifier holder.
controlId	When the action parameter (see above) is grant control, this parameter is the control identifier assigned by the sending ASCE (see below).
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

When an ASCE wishes to obtain the control identifier, it shall send a request control ControlPDU to all ASCEs. This ASPDU is sent to all ASCEs (as are all ControlPDUs), so that, even though ASCEs track the ASCE currently holding the control identifier, they may detect situations where the control identifier is in the process of moving, or ASCEs holding the control identifier become inactive or leave the conference.

On receipt of a request control ControlPDU, the ASCE holding the control identifier shall normally (but see below) send a grant control ControlPDU with the current control identifier value and the MCS user ID of the ASCE to which control is being granted to all ASCEs. This ASPDU is sent to all ASCEs so that all active ASCEs can track the current control identifier owner. Where an ASCE receives a request control ControlPDU when it is not holding the control identifier, it discards the ASPDU.

There are some situations where unconditionally granting control may not be feasible. For example, on some terminal types, certain window manager functions (such as window dragging and/or sizing a local window) need to be explicitly terminated. But if the local ASCE is in cooperating mode and loses control (and hence loses the right to provide any input), it cannot complete the window manager operation, which leaves the operation active and the peer ASCE unable to complete it (because it cannot provide input to a local window on this ASCE). Where such a situation applies, an ASCE holding the control identifier may respond to a request control ControlPDU with a grant control ControlPDU with the current control identifier value and its own MCS User ID (i.e., it grants control to itself). As this is sent to all ASCEs, peer ASCEs, including the requester, treat it as an ordinary exchange of control, and the local ASCE retains control until the problematic operation is complete.

Where an ASCE moves from cooperating to detached mode, it shall send a detach ControlPDU to all ASCEs. Similarly, where an ASCE moves from detached to cooperating mode, it shall send a cooperate ControlPDU to all ASCEs. Changing from cooperating to detached and vice versa is independent of ownership of the control identifier, and an ASCE holding the control identifier may move from cooperating to detached and back to cooperating mode without granting the control identifier if no peer ASCE requests it in the interim.

8.12.1 Control identifiers

The control identifier is a single value visible to all ASCEs within the conference, in the range 0..0x80000000.

The initial control identifier value in a conference is always zero. The initial control identifier holder in a conference is the ASCE whose share identifier is the highest during ASCE activation (see clause 8.4).

Thereafter, the control identifier value changes when an ASCE fails to receive a grant control response to a request control ControlPDU (within a reasonable time) or the control identifier holder becomes inactive or leaves the conference. Where either is the case, each detecting ASCE generates a new control identifier by incrementing the last known control identifier value by its MCS user ID and sends a grant control ControlPDU referencing itself as the owning ASCE (i.e., it advertises itself as the new control identifier holder). As multiple ASCEs may have detected the problem, this causes a control identifier race, which is won by the ASCE with the highest control identifier value. This creates a requirement that, at all times, if an ASCE receives a grant control ControlPDU with a higher control identifier value than the last control identifier value known to that ASCE, it shall recognize the higher value as the new control identifier value and the sending ASCE as the new control identifier holder.

When an ASCE detects that a new ASCE has become active, it shall advertise its control state as follows:

- If the ASCE is detached, it shall send a detach ControlPDU.
- If the ASCE is cooperating, it shall send a cooperate ControlPDU.
- If the ASCE holds the control identifier, it shall send a grant control ControlPDU referencing itself as the control identifier holder.

The net of the above is that new joiners (and existing ASCEs) receive a refresh of each ASCE's control state plus information on the holder of the control identifier. See clause 8.4 for further information on ASCE activation and clause 8.6 for further information on synchronization.

8.12.2 Interaction with conducted mode

Conducted mode operation (see clause 8.19) interacts with the AS control protocol as follows.

When a conference enters conducted mode, all ASCEs shall send a cooperate ControlPDU, the ASCE on the conducting node shall send a request control ControlPDU and the ASCE holding the control identifier shall respond with a grant control ControlPDU. That is, all ASCEs enter cooperating mode and the conducting node acquires control – all other ASCEs are viewing.

When Conductorship moves from one node to another, the new conducting node shall send a request control ControlPDU and the ASCE holding the control identifier (i.e., the previous conducting node) shall respond with a grant control ControlPDU. That is, control follows Conductorship.

When the conference exits conducted mode, all ASCEs remain in cooperating mode and the last conducting node retains the control identifier, but ASCEs are once again free to request control and to switch between cooperating and detached modes. That is, the full AS control protocol is reinstated.

8.13 Mediated control

The AS control protocol described in clause 8.12 provides a reasonable set of core control facilities. But it does not provide facilities such as the explicit passing of control to a specific ASCE or the ability for ASCEs to conditionally or unconditionally deny control requests.

The AS mediated control protocol builds on the core control protocol to provide additional, more conditional, control facilities. The mediated control protocol is negotiable and is supported only where the negotiated capabilities enable it (i.e., in legacy mode, where the negotiated Control.controlFlags capability allow mediated control bit flag is set; in base mode, where the

negotiated Control.mediatedControlFlag capability is TRUE). See clause 8.2.10 for further information on the control capability set.

The mediated control protocol is implemented by a set of request and response messages, which build upon and mediate the effect of the core control protocol. Where the explanation of a mediated control protocol facility or message exchange in this clause requires that an ASCE initiate a core control protocol action and/or state change, then it is shown as follows:

- Core (request control): The ASCE sends a request control ControlPDU to take control.
- Core (detach): The ASCE sends a detach ControlPDU to notify peer ASCEs that it has entered detached mode.

When an ASCE wishes to send a mediated control request or response, it shall send a MediatedControlPDU to one or all ASCEs in the conference in the manner indicated in Table 6-3. The content of the MediatedControlPDU is shown in Table 8-49.

Table 8-49 – MediatedControlPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
action	This parameter identifies the particular MediatedControlPDU action. The allowable actions are as follows: • Take control request. • Pass control request. • Detach request. • Confirm take response. • Deny take response. • Confirm detach response. • Deny detach response. • Deny pass response. • Remote detach request. • Deny remote detach response. See below for further information.
passControlFlag	This parameter indicates whether this MediatedControlPDU is part of a pass control sequence (see below). Where this parameter is part of a pass control sequence, the value is TRUE. On all other MediatedControlPDUs, this parameter shall be FALSE.
sendingReference	This parameter is a message reference used to correlate requests and responses. Where this MediatedControlPDU is a request (see the action parameter above), this is the reference allocated by the sending ASCE. Where this MediatedControlPDU is a response, this is the reference from the corresponding request.
originatorReference	This parameter is a message reference used to correlate requests and responses. Where this MediatedControlPDU is a take control request arising from a pass control request (see below), this parameter is the reference from the originating pass control request. Where this MediatedControlPDU is a response, this is the reference from the corresponding request.

Table 8-49 – MediatedControlPDU

Parameter	Description
originatorID	This parameter is a MCS user ID. Where this MediatedControlPDU is a request, this is the MCS user ID of the sending ASCE. Where this MediatedControlPDU is a response, this is the MCS user ID from the corresponding request.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.13.1 Taking control

When an ASCE wishes to take control using the mediated control protocol, its actions depend on the negotiated Control.controlInterest capability value. Where the negotiated value is always, the taking of control is unmediated and the ASCE initiates the core (request control) action to take control. See clause 8.2.10 for further information on the control capability set. Where the negotiated value is never, one or more peer ASCEs will not permit the taking of control and the ASCE cannot do so. Where the negotiated value is confirm, one or more peer ASCEs require that the taking of control requires confirmation by those peer ASCEs and the ASCE sends a take control request MediatedControlPDU to all ASCEs.

On receipt of a take control request MediatedControlPDU, an ASCE's response depends on its local Control.controlInterest capability value (which should not be never – or the request should not have been issued). Where the local value is always, the ASCE responds with a confirm take response MediatedControlPDU to the requesting ASCE. Where the local value is confirm, the ASCE utilizes a purely local mechanism (such as interacting with the local end-user) to determine whether to allow the requesting ASCE to take control and then responds with either a confirm take response or deny take response MediatedControlPDU to the requesting ASCE accordingly.

On receipt of confirm take response MediatedControlPDUs from all peer ASCEs (i.e., unanimous consent), the requesting ASCE initiates the core (request control) action to take control. However, if it receives one or more deny take response MediatedControlPDUs, then it abandons the attempt to take control.

8.13.2 Passing control

When an ASCE wishes to pass control to a specific peer ASCE using the mediated control protocol, its actions depend on its local Control.controlInterest capability value. Where the local value is never (i.e., this ASCE never gives up control), it should not attempt to pass control. Where the local value is always or confirm, it sends a pass control request MediatedControlPDU to the particular ASCE.

On receipt of a pass control request MediatedControlPDU, an ASCE utilizes a purely local mechanism (such as interacting with the local end-user) to determine whether to accept control from the requesting ASCE. If it determines not to accept the pass request, it responds with deny pass response MediatedControlPDU to the requesting ASCE. If it determines to accept the pass request, it sends a take control request to the requesting ASCE, which should in turn result in receipt of a confirm take response MediatedControlPDUs from the requesting ASCE, whereupon it can initiate the core (request control) action to take control.

Note that the take control request MediatedControlPDU is here sent to a single peer ASCE in response to a pass control request MediatedControlPDU, whereas in clause 8.13.1 above, it is sent to all peer ASCEs to conditionally take control. To distinguish between the two cases, all

MediatedControlPDUs used as part of the pass control sequence (i.e., pass control request, take control request to one ASCE and deny pass response) have the passControlFlag parameter set to TRUE – it is FALSE in all other MediatedControlPDUs.

8.13.3 Detaching

When an ASCE wishes to detach using the mediated control protocol, its actions depend on the negotiated Control.detachInterest capability value. Where the negotiated value is always, detaching is unmediated and the ASCE initiates the core (detach) action to detach. Where the negotiated value is never, one or more peer ASCEs will not permit ASCEs to detach and the ASCE cannot do so. Where the negotiated value is confirm, one or more peer ASCEs require that detaching requires confirmation by those peer ASCEs and the ASCE sends a detach request MediatedControlPDU to all ASCEs.

On receipt of a detach request MediatedControlPDU, an ASCE's response depends on its local Control.detachInterest capability value (which should not be never – or the detach request should not have been issued). Where the local value is always, the ASCE responds with a confirm detach response MediatedControlPDU to the requesting ASCE. Where the local value is confirm, the ASCE utilizes a purely local mechanism (such as interacting with the local end-user) to determine whether to allow the requesting ASCE to take control and then responds with either a confirm detach response or deny detach response MediatedControlPDU to the requesting ASCE accordingly.

On receipt of confirm detach response MediatedControlPDUs from all peer ASCEs (i.e., unanimous consent), the requesting ASCE initiates the core (detach) action to detach. However, if it receives one or more deny detach response MediatedControlPDUs, then it abandons the attempt to detach.

8.13.4 Remote detach

When an ASCE wishes to detach a peer ASCE, its actions depend on the Control.remoteDetachFlag capability value for the peer ASCE. Where the value is FALSE, the peer ASCE does not allow remote detach and the ASCE abandons the attempt. Where the value is TRUE, the ASCE sends a remote detach request MediatedControlPDU to the particular ASCE.

On receipt of a remote detach request MediatedControlPDU, an ASCE attempts to start the detach process described in clause 8.13.3. If it cannot initiate the detach attempt (because one or more peer ASCEs will not permit ASCEs to detach), it responds with a deny remote detach response MediatedControlPDU to the requesting ASCE. If it can initiate the detach attempt, then it proceeds as in clause 8.13.3.

MediatedControlPDUs are either sent to all peer ASCEs or to specific peer ASCEs depending on the message type. Table 8-50 summarizes the sending characteristics for MediatedControlPDU requests and responses.

Table 8-50 – MediatedControlPDU MCS channels

Request/response	Target
Take control request (Note)	All ASCEs
Take control request (Note)	Peer ASCE
Pass control request	Peer ASCE
Detach request	All ASCEs
Confirm take response	Peer ASCE
Deny take response	Peer ASCE
Confirm detach response	Peer ASCE

Table 8-50 – MediatedControlPDU MCS channels

Request/response	Target
Deny detach response	Peer ASCE
Deny pass response	Peer ASCE
Remote detach request	Peer ASCE
Deny remote detach response	Peer ASCE
NOTE – The two take control request variants are used in taking and passing control respectively (see clauses 8.13.1 and 8.13.2 above).	

8.14 Pointers

When the local pointer shape changes, or when an application programmatically changes the local pointer position, an ASCE shall send information about the local pointer shape and/or position to all ASCEs within the conference by sending PointerPDUs in the manner indicated in Table 6-3.

PointerPDUs supplying a new pointer shape (as opposed to updating the pointer position or referencing a previously cached pointer) are one of the following three types:

- A pre-defined system pointer value, which is either the null pointer or the default pointer. The null pointer should be sent when the local pointer is not displayed. The default pointer should be sent when the local pointer is not over a hosted window or not captured by a hosted window.
- A monochrome pointer definition. This monochrome pointer may be used where a default pointer is not appropriate (see above), a local pointer is visible and colour pointer support has been disabled during capabilities negotiation.
- A colour pointer definition. This colour pointer may be used where a default pointer is not appropriate (see above), a local pointer is visible and colour pointer support has been enabled during capabilities negotiation.

How an ASCE displays received pointer information is dependent on the current control state and conductorship (see clauses 8.12, 8.13 and 8.19), the current pointer position and the display capabilities of the local terminal. For example, a receiving ASCE might:

- display the local pointer shape when cooperating and in control;
- display the pointer corresponding to the hosting ASCE when cooperating and viewing;
- display a modified version of one or more hosting ASCE's pointers when detached and the particular pointer(s) are over the corresponding shadow windows.

8.14.1 System pointers

When the local pointer becomes invisible (i.e., it is not displayed on the local terminal), an ASCE shall send a PointerPDU (system) with a systemPointerType of pointerNull to all other ASCEs within the conference. On receipt of a PointerPDU (system) with a systemPointerType of pointerNull, an ASCE shall set its current colour pointer for that host to a special null pointer.

When the local pointer is not over a hosted window or not captured by a hosted window, an ASCE shall send a PointerPDU (system) with a systemPointerType of pointerDefault to all other ASCEs within the conference. On receipt of a PointerPDU (system) with a systemPointerType of default pointerDefault, an ASCE shall set its current colour pointer for that host to a special default pointer.

For both cases, while the receiving ASCE is required to track the current pointer for each other hosting ASCE, how it displays that pointer is a local matter. See Table 8-51.

Table 8-51 – PointerPDU (system)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
systemPointerType	This parameter identifies the pointer to be used. The allowable values are pointerDefault or pointerNull.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.14.2 Monochrome pointers

An ASCE is not required to support colour pointers (see clause 8.14.3) and may represent all non-system pointers as monochrome pointers (even where they are displayed locally in colour), provided it can implement a suitable local conversion. However, colour pointers are widely used on common terminals, and providing monochrome-only pointer support may result in a significant reduction in user acceptability.

Where the local pointer changes to a pointer that is, or may be represented as, a monochrome pointer, an ASCE shall send a PointerPDU (mono) containing the new pointer's monochrome pointer definition. A receiving ASCE is only required to remember the last monochrome pointer definition from each other hosting ASCE which then becomes the current monochrome pointer for that host.

While a receiving ASCE is required to track the current monochrome pointer for each other hosting ASCE, how it displays that pointer is a local matter.

Monochrome pointers are not cached. A sending ASCE is required to send a new PointerPDU (mono) for each monochrome pointer change and a receiving ASCE is only required to remember the last monochrome pointer definition from each other hosting ASCE.

Monochrome pointer data represents the pointer as a pair of monochrome AND and XOR masks, where the pointer can be drawn by ANDing the AND mask and then XORing the XOR mask to the display, allowing for the hot spot (see below). This is a common representation, which can be readily mapped to standard local functions on most common terminal types.

Both masks are a series of monochrome pixel rows (i.e., 1 bit-per-pixel), where row zero starts at the highest pixel Y coordinate. That is, row 0 starts at (top, left). Within a row, pixel values are packed into octets, starting from the left-most pixel. Each octet contains eight pixels, with the leftmost pixel in the most significant bit. Each row of pointer data is padded to a two-octet boundary.

The monochrome pointer hot spot defines a point within the pointer, which corresponds to the position on the display where the pointer should be drawn. For example, if the monochrome pointer hot spot is at point (3,4) within the pointer and the local pointer position is at point (50,50), then the (top, left) pixel (i.e., 0,0) of the pointer definition is drawn at (47,46) and the hot spot pixel is drawn at (50,50). See Table 8-52.

Table 8-52 – PointerPDU (mono)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
hotSpotX	This parameter is the pointer hot spot X coordinate in pixels relative to (top, left) of the pointer definition.
hotSpotY	This parameter is the pointer hot spot Y coordinate in pixels relative to (top, left) of the pointer definition.
width	This parameter is the width of the pointer in pixels.
height	This parameter is the height of the pointer in pixels.
monoPointer	This is a monochrome pointer definition, consisting of a 1 bits-per-pixel XOR mask, followed by a 1 bits-per-pixel AND mask.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.14.3 Colour pointers

Colour pointer support is optional. An ASCE may only send PointerPDU (colour) and PointerPDU (cached) ASPDUs where the negotiated Pointer.colorPointerFlag capability is TRUE. Where colour pointers are not supported, an ASCE is required to send all colour pointer changes as monochrome pointers (see clause 8.14.2). It is recommended that ASCEs support colour pointers.

Colour pointers may be cached, depending on the result of pointer capabilities negotiation. If an ASCE supports colour pointer caching, it shall set its advertised Pointer.pointerCacheSize capability to the number of cached colour pointer entries it is prepared to support for each other hosting ASCE (see clause 8.2.11).

Where colour pointers are supported and colour pointer caching is enabled after capabilities negotiation (i.e., the negotiated Pointer.pointerCacheSize is greater than one), a sending ASCE may allocate a colour pointer definition to a specific pointer cache entry on receiving ASCEs using a PointerPDU (colour) and then subsequently reuse that cached entry by sending a PointerPDU (cached) referencing that entry. It is the sending ASCE's responsibility to manage the colour pointer cache entry space.

On receipt of a PointerPDU (colour) from a particular hosting ASCE referencing a colour pointer cache entry, an ASCE shall place the colour pointer definition in the cache entry for that host and set the supplied colour pointer definition as the current colour pointer for that host. On receipt of PointerPDU (cached) from a particular hosting ASCE referencing a colour pointer cache entry, an ASCE shall set the referenced cached colour pointer definition as the current colour pointer for that host.

Where colour pointer caching is disabled, an ASCE shall not send the PointerPDU (cached) ASPDU and may only send information on local pointer changes via a series of PointerPDUs (colour). A receiving ASCE with colour pointer caching disabled is only required to remember the last colour pointer definition from each other hosting ASCE which is always the current colour pointer for that host.

While a receiving ASCE is required to track the current colour pointer for each other hosting ASCE, how it displays that pointer is a local matter.

Colour pointer data represents the pointer as a series of rows, where row zero starts at the lowest pixel Y coordinate. That is, row 0 starts at (bottom, left). Within a row, pixel values are packed into octets, starting from the left-most pixel. For the 1 bits-per-pixel AND mask data, each octet contains

eight pixels, with the leftmost pixel in the most significant bit. For the 24 bits-per-pixel XOR mask data, each octet triplet contains one pixel of RGB colour information.

Colour pointer data represents the pointer as a pair of AND and XOR masks, where the pointer can be drawn by ANDing the AND mask and then XORing the XOR mask to the display, allowing for the hot spot (see below). The AND mask is monochrome and the XOR mask is colour. This is a common representation, which can be readily mapped to standard local functions on most common terminal types.

The AND mask is a series of monochrome pixel rows (i.e., 1 bit-per-pixel), where row zero starts at the highest pixel Y coordinate. That is, row 0 starts at (top, left). Within a row, pixel values are packed into octets, starting from the left-most pixel. Each octet contains eight pixels, with the left-most pixel in the most significant bit. Each row of pointer data is padded to a two-octet boundary.

The XOR mask is 24 bits-per-pixel, where row zero starts at the highest pixel Y coordinate. That is, row 0 starts at (top, left). Within a row, each triplet of octets contains one pixel of RGB colour information. Within an RGB triplet, the first octet is a blue value in the range 0..255, the second octet is a green value in the range 0..255 and the third octet is a red value in the range 0..255. Octets are tightly packed – there are no pad octets between adjacent RGB values. Each row of pointer data is padded to a four-octet boundary.

The colour pointer hot spot defines a point within the pointer, which corresponds to the position on the display where the pointer should be drawn. For example, if the colour pointer hot spot is at point (3,4) within the pointer and the local pointer position is at point (50,50), then the (top, left) pixel (i.e., 0,0) of the pointer definition is drawn at (47,46) and the hot spot pixel is drawn at (50,50). See Tables 8-53 and 8-54.

Table 8-53 – PointerPDU (colour)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
cacheIndex	This parameter specifies the pointer cache entry to use for this pointer. Allowable values are in the range zero to one less than the negotiated Pointer.pointerCacheSize capability value. See clause 8.2.11 for further information on the pointer capability set.
hotSpotX	This parameter is the pointer hot spot X coordinate in pixels relative to (top, left) of the pointer definition.
hotSpotY	This parameter is the pointer hot spot Y coordinate in pixels relative to (top, left) of the pointer definition.
width	This parameter is the width of the pointer in pixels.
height	This parameter is the height of the pointer in pixels.
colorPointer	This is the colour pointer definition, consisting of a 24 bits-per-pixel XOR mask, followed by a 1 bit-per-pixel AND mask.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

Table 8-54 – PointerPDU (cached)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
cacheIndex	This parameter specifies which cached pointer to use. Allowable values are in the range zero to one less than the negotiated Pointer.pointerCacheSize capability value. See clause 8.2.11 for further information on the pointer capability set.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.14.4 Pointer position updates

Normally, pointer position information is conveyed by input events driven by user pointing device activity. See clause 8.18 for further information on pointing device movement. However, some terminals allow applications to programmatically update the local pointer position.

On such terminals, where an application programmatically updates the local pointer position, an ASCE shall send a PointerPDU (PointerPosition) containing the new virtual desktop pointer position. On receipt of a PointerPDU (PointerPosition) containing a new pointer position, a receiving ASCE updates the current pointer position for that host.

While a receiving ASCE is required to track the current pointer position for each other hosting ASCE, how it processes that pointer position information is a local matter. See Table 8-55.

Table 8-55 – PointerPDU (PointerPosition)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
pointerX	This parameter is the X virtual desktop coordinate of the new pointer position.
pointerY	This parameter is the Y virtual desktop coordinate of the new pointer position.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.15 Palette updates

An ASCE sends palette updates to all ASCEs within the conference by sending an UpdatePDU containing a palette in the manner indicated in Table 6-3. The content of the UpdatePDU containing a palette is shown in Table 8-56.

For example, if a local application changes a local palette and then draws a bitmap with respect to that palette, then the ASCE shall ensure that it sends a palette UpdatePDU before the bitmap UpdatePDU containing that bitmap data. Whenever an ASCE sends a new palette, it shall ensure that all hosted windows on its local terminal are redrawn, to ensure that receiving ASCEs can redraw any already received areas of the hosted windows with reference to the new palette.

The AS protocol supports colour depths of 1, 4 and 8 bits-per-pixel (see clause 8.2.4). An ASCE shall not send palette updates where the sendingBitsPerPixel is 1. For this case, the AS protocol defines palette indices 0 and 1 as colour values black and white respectively.

A palette contains 16 or 256 RGB colour values. The arrangement of colour values in the palette is significant and represents a sequence of palette indices from 0..15 or 0..255, depending on the sendingBitsPerPixel. In the base mode of the AS protocol, the palette may also contain optional colour accuracy information.

On receipt of an UpdatePDU containing a palette, an ASCE shall use that palette to interpret subsequent bitmap data pixel values. For example, in incoming bitmap data at 8 bits-per-pixel, the receiving ASCE interprets a bitmap pixel containing the value 25 as referencing the 26th (allowing for indexing from zero) colour value in the last received palette from the sending ASCE. It shall not use the palette to interpret cached bitmap pixel values – which should be interpreted using the appropriate cached ColorTable (see clause 8.16.10).

Table 8-56 – UpdatePDU (palette)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
palette	This parameter is a list of colour values constituting the palette. In the base mode of the AS protocol, the palette may also contain optional colour accuracy information.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16 Order updates

An ASCE sends order updates to all ASCEs within the conference by sending an UpdatePDU containing orders in the manner indicated in Table 6-3. The content of the UpdatePDU containing order updates is shown in Table 8-57.

Table 8-57 – UpdatePDU (orders)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
orderList	This parameter is a list of orders.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

UpdatePDU orders may be of the following two types:

- Primary orders (see Table 8-58): Primary orders are drawing orders that may result in output on remote ASCEs, subject to ASCE clipping, window order and/or presentation.
- Secondary orders (see Table 8-59): Secondary orders provide ancillary information for subsequent use by primary orders. For example, secondary orders are used to populate remote ASCE bitmap and ColorTable caches prior to cache references by subsequent primary orders.

A single UpdatePDU containing orders may contain any mix of primary and secondary orders.

Table 8-58 – Primary orders

Order	Reference
Destination Blt	See Table 8-65
Pattern Blt	See Table 8-66
Screen Blt	See Table 8-67
Memory Blt	See Table 8-70
Memory three-way Blt	See Table 8-71
Text	See Table 8-72
Extended text	See Table 8-73
Frame	See Table 8-74
Rectangle	See Table 8-75
Opaque rectangle	See Table 8-76
Line	See Table 8-77
Desktop save	See Table 8-78
Desktop origin	See Table 8-79

Table 8-59 – Secondary orders

Order	Reference
Cache bitmap (compressed)	See Table 8-68
Cache bitmap (uncompressed)	See Table 8-68
Cache ColorTable	See Table 8-69
Color space	See Table 8-80

8.16.1 Primary orders

Primary orders shall contain the primary order header. The legacy mode primary order header (see Table 8-60) contains explicit order encoding information, whereas the base mode primary order header (see Table 8-61) does not. See clause 8.16.3 for further information on order encoding.

An ASCE shall only send an UpdatePDU containing primary orders where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and where the corresponding order is supported (i.e., in legacy mode, the corresponding negotiated Order.ordersupport capability entry is non-zero; in base mode, the order's corresponding negotiated order level capability is non-zero). There may also be further order-specific restrictions, which are documented, where applicable, in the description of the particular primary order.

An ASCE may supply bounds coordinates on primary orders. Bounds coordinates define a clipping rectangle for the order on which they are present. Where bounds coordinates are supplied on an order (after any required order decoding), the receiving ASCE shall draw the order clipped to the bounds and the virtual desktop. If bounds coordinates are not supplied on an order (after any required order decoding), the receiving ASCE shall not clip the order, other than to the virtual desktop. Where bounds coordinates are supplied (prior to order encoding) in the legacy mode of the AS protocol, the ASCE should set the bounds control flag in the order header control flags.

Table 8-60 – Primary order header (legacy mode)

Parameter	Description
controlFlags	This parameter indicates the order encoding options applicable to the order that follows. See Table 8-63 for further information on order header control flags.
orderType (optional)	This parameter is present where the controlFlags parameter has the type change flag set. Where present, it specifies the type of order that follows. See Table 8-58 for a summary of allowable values.
encodingFlags	This parameter is 1..3 octets that indicate the number of encodable parameters that are present in the order. See Table 8-64 for further information on the number of encodable parameters that are allowable per primary order.
boundsFlags (optional) v	This parameter is present where the controlFlags parameter has the bounds flag set. Where present, it is a set of bit flags indicating which bounds coordinates are present in the bounds parameter (see below) and in which format, subject to bounds coordinates encoding. Defined bit flag values are as follows: • Absolute left bounds coordinate present. • Absolute top bounds coordinate present. • Absolute right bounds coordinate present. • Absolute bottom bounds coordinate present. • Delta left bounds coordinate present. • Delta top bounds coordinate present. • Delta right bounds coordinate present. • Delta bottom bounds coordinate present. See clause 8.16.3.2 for further information on bounds coordinates encoding.
bounds (optional)	This parameter is a set of zero to four absolute or delta bounds coordinates, depending on the bit flags set in the boundsFlags parameter (see above).

Table 8-61 – Primary order header (base mode)

Parameter	Description
bounds (optional)	This parameter is a set of zero to four absolute or delta bounds coordinates.

8.16.2 Secondary orders

In the legacy mode of the AS protocol, secondary orders shall contain the secondary order header described in Table 8-62. In the base mode of the AS protocol, secondary orders do not contain an explicit header.

An ASCE shall only send an UpdatePDU containing secondary orders where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and, in base mode, where the corresponding order is supported (i.e., the order's corresponding negotiated order level capability is

non-zero). There may also be further order-specific restrictions, which are documented, where applicable in the description of the particular secondary order.

Table 8-62 – Secondary order header (legacy mode)

Parameter	Description
controlFlags	This parameter indicates the order encoding options applicable to the order that follows. See Table 8-63 for further information on order header control flags.
extraFlags	This parameter is a set of bit flags indicating whether the order that follows is primary or secondary. The only defined bit flag value is secondary, which shall be set.

8.16.3 Order encoding

The AS protocol uses order encoding to minimize the number of parameters within an order based on difference comparisons with previous orders. There are a number of defined order encoding methods, which, where applicable and where applied, shall be serially applied in the following order:

1) Type encoding: See clause 8.16.3.1.

2) Bounds coordinates encoding: See clause 8.16.3.2.

3) Parameter encoding: See clause 8.16.3.3.

4) Coordinate encoding: See clause 8.16.3.4.

Order encoding may only be applied to primary orders (i.e., secondary orders are not encoded). In the legacy mode of the AS protocol, order encoding information is conveyed via the controlFlags, encodingFlags and boundsFlags in the primary order header (see Table 8-60). In the base mode of the AS protocol, order encoding information is implicitly conveyed by the presence or absence of optional order parameters.

Both modes require that sending and receiving ASCEs monitor the order stream sent in UpdatePDU (orders) ASPDUs to maintain an encoding state and potentially multiple decoding states (when receiving orders from multiple hosting ASCEs), such that sending ASCEs may omit unchanged, or may supply deltas for small changes in, order parameters, while receiving ASCEs can correctly reconstruct the original unencoded order stream.

In the legacy mode of the AS protocol, all orders contain an initial set of control flags, which indicates how the rest of the order should be interpreted. See Table 8-63 for further information on the allowable control flags. For legacy mode primary orders, an ASCE shall always set the standard encoding control flag, shall not set the secondary control flag and may set any of the other allowable control flags as required by order encoding. For legacy mode secondary orders, an ASCE shall always set the standard encoding and secondary control flags and shall not set any of the other allowable control flags.

Table 8-63 – Order header control flags (legacy mode)

Control flag	Description
Standard encoding	This flag indicates that the order that follows conforms to the standard encoding for this Recommendation. This flag shall be set in all orders.
Secondary	This flag (if set) indicates that the order that follows is a secondary order. This flag shall only be set on secondary orders.

Table 8-63 – Order header control flags (legacy mode)

Control flag	Description
Bounds	This flag (if set) indicates that the order that follows includes bounds. This flag shall only be set on primary orders. See clause 8.16.3.2 for further information on bounds coordinates encoding.
Type change	This flag (if set) indicates that the order that follows is of a different type to the previous order. This flag shall only be set in primary orders. See clause 8.16.3.1 for further information on type encoding.
Delta coordinates	This flag (if set) indicates that the order that follows uses delta coordinates (rather than absolute coordinates). This flag shall only be set in primary orders. See clause 8.16.3.4 for further information on coordinate encoding.

8.16.3.1 Type encoding

In the legacy mode of the AS protocol, where an order is the same type as the previous order, then an ASCE may clear the type change bit flag in the controlFlags parameter and omit the orderType parameter in the encoded order. Where a primary order is not of the same type as the previous order, an ASCE shall set the type change bit flag in the controlFlags parameter and supply the orderType parameter.

Type encoding is applied for the continuous order stream and spans UpdatePDU (orders) ASPDUs.

Type encoding is not supported in the base mode of the AS protocol.

8.16.3.2 Bounds coordinates encoding

Where one or more of the bounds coordinates of an order are the same as the corresponding bounds coordinate in the previous order (which need not be of the same type), an ASCE may omit the unchanged bounds coordinates. Bounds order encoding is applied to individual bounds coordinates with respect to the same bounds coordinate in the previous order. For example, where both order^{i} and order^{i+1} have a left bounds coordinate value of 100, then the left bound may be omitted in the encoded order^{i+1}. Conversely, where order^{i}'s (left, top) bounds are (110, 100) and order^{i+1}'s (left, top) bounds are (100, 110), neither the left nor top bounds coordinate may be omitted.

Bounds coordinate encoding is applied for the continuous order stream and spans UpdatePDU (orders) ASPDUs.

Where a bounds coordinate is changed from the previous order, it is then subject to coordinate encoding (see clause 8.16.3.4). This means that a particular order may omit some bounds coordinates, and may contain a mix of delta and absolute bounds coordinates for the remaining bounds coordinates.

In the base mode of the AS protocol, the presence or absence of bounds coordinates and, where present, whether they are represented by absolute or delta coordinate bits is conveyed by the ASN.1 encoding (see clause 9).

In the legacy mode of the AS protocol:

- where some or all bounds coordinates are omitted, an ASCE shall clear both the corresponding absolute and delta bounds bit flags in the boundsFlag parameter in the primary order header (see Table 8-60);
- where one or more bounds coordinates are changed from the previous order and the change is in the range –128..+127 pixels, an ASCE may supply single octet delta coordinates for those bounds coordinates and set the corresponding delta coordinate bits in the boundsFlag parameter;

- where one or more bounds coordinates are changed from the previous order and the change is not in the range –128.. +127 pixels, an ASCE shall supply two-octet absolute coordinates for those bounds coordinates and set the corresponding absolute coordinate bits in the boundsFlag parameter.

8.16.3.3 Parameter encoding

Where a parameter is the same as the corresponding parameter in the previous order of the same type, an ASCE may omit the unchanged parameter. For example, where both order^{i} and order^{i+n} (where n may be greater than one) are consecutive pattern Blt orders and both have a ROP3 parameter of 0xCC, then the ROP3 parameter may be omitted in the encoded order^{i+n}. See clause 8.16.5 for further information on pattern Blt orders and clause 8.16.20 for further information on three-way ROPs.

Parameter encoding is applied with respect to the same parameter in orders of the same type across the continuous order stream and spans UpdatePDU (orders) ASPDUs.

In the base mode of the AS protocol, the presence or absence of parameters is conveyed by the ASN.1 encoding (see clause 9).

In the legacy mode of the AS protocol, the presence or absence of a particular parameter is indicated by the corresponding bit flag in the encodingFlags parameter in the primary order header (see Table 8-60). For each order:

- the first parameter is parameter zero;
- the number of octets required for the encodingFlags parameter is (number of encodable parameters +7) div 8;
- the bit flag for parameter N is represented by encodingFlags octet (N div 8), bit flag (N mod 8), where the first encodingFlags octet is octet zero and the first bit within an octet is the least significant bit.

Common parameters such as brushes and pens are encoded according to their constituent parameters. For example, the brush parameter in the pattern Blt order (see Table 8-66), which is documented as a single parameter (for the purposes of clarity), contributes five encodable parameters to the total of twelve encodable parameters for the pattern Blt order. Such grouped parameters are marked as (group) in the respective order descriptions.

For example, in the parameter encoding for the pattern Blt order (see clause 8.16.5)

- the foregroundColor parameter is parameter 6 and is represented by octet 0, bit 6;
- the brush hatch parameter is parameter 10 and is represented by octet 1, bit 2.

Table 8-64 summarizes the number of encodable parameters per order.

Table 8-64 – Primary orders: Encodable parameters

Order	Number of encodable parameters	Number of encodingFlags octets	Reference
Destination Blt	5	1	See Table 8-65
Pattern Blt	12	2	See Table 8-66
Screen Blt	7	1	See Table 8-67
Memory Blt	9	2	See Table 8-70
Memory three-ay Blt	17	3	See Table 8-71
Text	14	2	See Table 8-72

Table 8-64 – Primary orders: Encodable parameters

Order	Number of encodable parameters	Number of encodingFlags octets	Reference
Extended text	20	3	See Table 8-73
Frame	14	2	See Table 8-74
Rectangle	17	3	See Table 8-75
Opaque rectangle	5	1	See Table 8-76
Line	10	2	See Table 8-77
Desktop save	6	1	See Table 8-78
Desktop origin	2	1	See Table 8-79

8.16.3.4 Coordinate encoding

After the application of any parameter encoding (see clause 8.16.3.3), an ASCE may apply coordinate encoding to any remaining coordinate parameters.

Where the change in coordinate parameters with respect to the same coordinate parameters in the previous order of the same type, is in the range –128..+127 pixels, an ASCE may supply single octet delta coordinates for coordinate parameters.

In the legacy mode of the AS protocol, coordinate encoding requires that all not previously encoded coordinates in an order can be represented as delta coordinates. Therefore, where the change in one or more not previously encoded coordinates is outside the range –128..+127 pixels, an ASCE shall use two-octet absolute coordinates for all coordinate parameters. In the base mode of the AS protocol, there is no such restriction, and an ASCE may supply a mix of delta and absolute coordinates, depending solely on the change in a particular not previously encoded coordinate parameter with respect to the same parameter in the previous order of the same type.

In the legacy mode of the AS protocol, an ASCE shall set or clear the delta coordinates bit flag in the controlFlags parameter, to indicate whether all coordinates are conveyed as delta or absolute coordinates. In the base mode of the AS protocol, the presence or absence of coordinates, and whether they are delta or absolute, is conveyed by the ASN.1 encoding (see clause 9).

Coordinate encoding is applied with respect to (not previously encoded) coordinate parameters in an order with respect to the same coordinate parameters in the previous order of the same type across the continuous order stream and spans UpdatePDU (orders) ASPDUs.

Coordinate parameters that are eligible for coordinate encoding are marked as (coordinate) in the order parameter descriptions and the grouped parameter definitions.

8.16.4 Destination Blt

An ASCE shall only send a destination Blt order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode the negotiated Order.receiveOrdersFlag capability is TRUE) and where the destination Blt order is supported (i.e., in legacy mode the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.DestinationBltLevel capability is non-zero).

An ASCE receiving the destination Blt order performs the ROP3 parameter raster operation on the destination rectangle on the virtual desktop, subject to any bounds clipping. See Table 8-65.

Table 8-65 – Destination Blt order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this destination Blt.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this destination Blt.
destWidth (coordinate)	This parameter is the width in pixels of the destination rectangle for this destination Blt.
destHeight (Coordinate)	This parameter is the height in pixels of the destination rectangle for this destination Blt.
ROP3	This parameter is the three-way ROP to use for this destination Blt. For destination Blt orders, the three-way ROP shall reference a destination and shall not reference a source or pattern. See clause 8.16.20 for further information on three-way ROPs.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.5 Pattern Blt

An ASCE shall only send a pattern Blt order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and where the pattern Blt order is supported (i.e., in legacy mode, the negotiated corresponding Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.PatternBltLevel capability is non-zero).

An ASCE receiving the pattern Blt order performs the ROP3 parameter raster operation on the brush and the destination rectangle on the virtual desktop, subject to any bounds clipping. See Table 8-66.

Table 8-66 – Pattern Blt order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this pattern Blt.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this pattern Blt.
destWidth (coordinate)	This parameter is the width in pixels of the destination rectangle for this pattern Blt.
destHeight (coordinate)	This parameter is the height in pixels of the destination rectangle for this pattern Blt.
ROP3	This parameter is the three-way ROP to use for this pattern Blt. For pattern Blt orders, the three-way ROP shall reference a pattern and shall not reference a source. See clause 8.16.20 for further information on three-way ROPs.
backgroundColor	This parameter is the background colour to use for this pattern Blt.

Table 8-66 – Pattern Blt order

Parameter	Description
foregroundColor	This parameter is the foreground colour to use for this pattern Blt.
brush (group)	This parameter is the brush to use for this pattern Blt. See clause 8.16.22 for further information on brushes.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.6 Screen Blt

An ASCE shall only send a screen Blt order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and where the screen Blt order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.ScreenBltLevel capability is non-zero).

An ASCE receiving the screen Blt order performs the ROP3 parameter raster operation on the source and destination rectangles on the virtual desktop, subject to any bounds clipping:

- The source rectangle is defined by the sourceX, sourceY parameters and the destWidth, destHeight parameters.
- The destination rectangle is defined by the destLeft, destTop parameters and the destWidth, destHeight parameters.
- The destWidth and destHeight parameters define the width and height of both the source and destination rectangles. This precludes any stretching.
- The source may overlap the destination.

See Table 8-67.

Table 8-67 – Screen Blt order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this screen Blt.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this screen Blt.
destWidth (coordinate)	This parameter is the width in pixels of the destination rectangle for this screen Blt.
destHeight (coordinate)	This parameter is the height in pixels of the destination rectangle for this screen Blt.
ROP3	This parameter is the three-way ROP to use for this screen Blt. For screen Blt orders, the three-way ROP shall reference a source and shall not reference a pattern. See clause 8.16.20 for further information on three-way ROPs.
sourceX (coordinate)	This parameter is the source X virtual desktop coordinate for this screen Blt.

Table 8-67 – Screen Blt order

Parameter	Description
sourceY (coordinate)	This parameter is the source Y virtual desktop coordinate for this screen Blt.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.7 Cache bitmap

An ASCE shall only send a cache bitmap order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and either of the memory Blt or memory three-way Blt orders are supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entries are non-zero; in base mode either of the negotiated Order.MemoryBltLevel or Order.MemoryThreeWayBltLevel capabilities are non-zero). In addition, in base mode the negotiated Order.CacheBitmapLevel capability shall be non-zero.

The sending ASCE shall further ensure that the cache bitmap order parameters reflect the outcome of bitmap and bitmap cache capabilities negotiation (see clauses 8.2.4 and 8.2.7), namely that:

- it shall only send a cache bitmap (compressed) order where the negotiated bitmap capability set values indicate that bitmap compression is supported;
- the bitmapBitsPerPixel parameter is equal to the negotiated sendingBitsPerPixel value;
- the cacheID value references a bitmap cache area where the negotiated value indicates at least one entry;
- the cacheIndex value within that bitmap cache area is less than the negotiated number of entries;
- the bitmap size (after any compression) fits into the negotiated maximum cell size for that bitmap cache area.

The sending ASCE is responsible for allocating the cacheID and cacheIndex parameters and, therefore, for populating and updating the receiving ASCE's bitmap cache. This requires that hosting ASCEs track bitmap cache usage based on previously sent cache bitmap orders.

An ASCE receiving a cache bitmap order from a particular hosting ASCE places the supplied bitmap into the cacheID bitmap cache area for that hosting ASCE at entry cacheIndex. The supplied bitmap replaces any existing cached bitmap in that slot. This requires that all active ASCEs maintain a separate bitmap cache for each other hosting ASCE. See Table 8-68.

Table 8-68 – Cache bitmap order

Parameter	Description
Secondary order header	The secondary order header is described in clause 8.16.2.
cacheID	This parameter indicates which bitmap cache should be used for this bitmap. Allowable values, depending on bitmap cache negotiation, are in the range 0..2. See clause 8.2.7 for further information on bitmap cache negotiation.
bitmapWidth	This parameter is the width in pixels of the bitmap.

Table 8-68 – Cache bitmap order

Parameter	Description
bitmapHeight	This parameter is the height in pixels of the bitmap.
bitmapBitsPerPixel	This parameter is the bits-per-pixel of the bitmap data. This parameter shall be the same as the negotiated Bitmap.sendingBitsPerPixel capability value. See clause 8.2.4 for further information.
cacheIndex	This parameter indicates which cache entry to use within the particular cache indicated by the cacheID parameter (see above). Allowable values depend on the outcome of bitmap cache capabilities negotiation of cache cell sizes for each cache. See clause 8.2.7 for further information on bitmap cache negotiation.
bitmapData	This parameter is the bitmap data to cache. The bitmap may be uncompressed (see clause 8.17.1) or compressed (see clause 8.17.2).
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.8 Cache ColorTable

An ASCE shall only send a cache ColorTable order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and either of the memory Blt or memory three-way Blt orders are supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entries are non-zero; in base mode, either the negotiated Order.MemoryBltLevel or Order.MemoryThreeWayBltLevel capabilities are non-zero). In addition, in base mode the negotiated Order.CacheColorTableLevel capability shall be non-zero.

The sending ASCE shall further ensure that the cache ColorTable order parameters reflect the outcome of bitmap and ColorTable cache capabilities negotiation (see clause 8.2.4 and clause 8.2.8), namely that:

- The number of entries in the ColorTable is based on the negotiated sendingBitsPerPixel value as follows:

  number of ColorTable entries = 2 to power of sendingBitsPerPixel

  (This gives an allowable number of ColorTable entries of 16 or 256).
- The cacheIndex value within the ColorTable cache is less than the negotiated number of entries.

An ASCE shall not send the cache ColorTable order where the Bitmap.sendingBitsPerPixel is 1. For this case, the AS protocol defines ColorTable indices 0 and 1 as colour values black and white, respectively. See clause 8.2.4 for further information on the bitmap capability set.

The sending ASCE is responsible for allocating the cacheIndex parameter and, therefore, for populating and updating the receiving ASCE's ColorTable cache. This requires that hosting ASCEs track ColorTable cache usage at all other active ASCEs, based on previously sent cache ColorTable orders.

An ASCE receiving the cache ColorTable order places the supplied ColorTable into its ColorTable cache area at entry cacheIndex. The supplied ColorTable replaces any existing cached ColorTable

in that slot. This requires that all active ASCEs maintain a separate ColorTable cache for each other hosting ASCE.

A ColorTable contains 16 or 256 RGB colour values. The arrangement of colour values in the ColorTable is significant and represents a sequence of ColorTable indices from 0..15 or 0..255, depending on the sendingBitsPerPixel. In the base mode of the AS protocol, the ColorTable may also contain optional colour accuracy information. See Table 8-69.

Table 8-69 – Cache ColorTable order

Parameter	Description
Secondary order header	The secondary order header is described in clause 8.16.2.
cacheIndex	This parameter indicates which cache entry to use within the ColorTable cache. Allowable values depend on the outcome of ColorTable cache capabilities negotiation of cache size. See clause 8.2.8 for further information on the ColorTable cache capability set.
colorTable	This parameter is a list of colour values comprising the ColorTable to be cached. In the base mode of the AS protocol, the ColorTable may also contain optional colour accuracy information.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.9 Memory Blt

An ASCE shall only send a memory Blt order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and where the memory Blt order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.MemoryBltLevel capability is non-zero).

The sending ASCE shall ensure that the bitmapCacheID, bitmapCacheIndex and colorTableCacheIndex parameters refer to a bitmap and ColorTable previously cached using the cache bitmap and cache ColorTable orders.

An ASCE receiving a memory Blt order performs the ROP3 parameter raster operation using the source rectangle in the cached bitmap and the destination rectangle on the virtual desktop, subject to any bounds clipping.

- The source rectangle is defined by the sourceX, sourceY parameters and the destWidth, destHeight parameters.
- The destination rectangle is defined by the destLeft, destTop parameters and the destWidth, destHeight parameters.
- The destWidth and destHeight parameters define the width and height of both the source and destination rectangles. This precludes any stretching.
- The source rectangle is completely within the cached bitmap dimensions. The receiving ASCE does not need to perform clipping of the source rectangle against the cached bitmap dimensions.
- The cached bitmap bits shall be interpreted using the referenced cached ColorTable.

See Table 8-70.

Table 8-70 – Memory Blt order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
colorTableCacheIndex	This parameter specifies the cached ColorTable to use for this memory Blt. See clause 8.16.8 for further information on ColorTable caching. For legacy mode memory Blt orders, this parameter and the bitmapCacheID parameter (see below) are both present where the appropriate bit flag is set in the encodingFlags parameter in the primary order header (Note).
bitmapCacheID	This parameter, when used in conjunction with the bitmapCacheIndex parameter (see below), specifies the cached bitmap to use for this memory Blt. See clause 8.16.7 for further information on bitmap caching. For legacy mode memory Blt orders, this parameter and the colorTableCacheIndex parameter (see above) are both present where the appropriate bit flag is set in the encodingFlags parameter in the primary order header (Note).
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this memory Blt.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this memory Blt.
destWidth (coordinate)	This parameter is the width in pixels of the destination rectangle for this memory Blt.
destHeight (coordinate)	This parameter is the height in pixels of the destination rectangle for this memory Blt.
ROP3	This parameter is the three-way ROP to use for this memory Blt. For memory Blt orders, the three-way ROP shall reference a source and shall not reference a pattern. See clause 8.16.20 for further information on three-way ROPs.
sourceX (coordinate)	This parameter is the source X coordinate in the referenced cached bitmap for this memory Blt.
sourceY (coordinate)	This parameter is the source Y coordinate in the referenced cached bitmap for this memory Blt.
bitmapCacheIndex	This parameter, when used in conjunction with the bitmapCacheID parameter (see above), specifies the cached bitmap to use for this memory Blt. See clause 8.16.7 for further information on bitmap caching.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.
NOTE – For legacy mode memory Blt orders, the colorTableCacheIndex and bitmapCacheID parameters are counted as one parameter in the primary order header encodingFlags. See clause 8.16.3.3 for further information.	

8.16.10 Memory three-way Blt

An ASCE shall only send a memory three-way Blt order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and where the memory three-way Blt order is

supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.MemoryThreeWayBltLevel capability is non-zero).

The sending ASCE shall ensure that the bitmapCacheID, bitmapCacheIndex and colorTableCacheIndex parameters refer to a bitmap and ColorTable previously cached using the cache bitmap and cache ColorTable orders.

An ASCE receiving a memory three-way Blt order performs the ROP3 parameter raster operation using the brush, the source rectangle in the cached bitmap and the destination rectangle on the virtual desktop, subject to any bounds clipping.

- The source rectangle is defined by the sourceX, sourceY parameters and the destWidth, destHeight parameters.
- The destination rectangle is defined by the destLeft, destTop parameters and the destWidth, destHeight parameters.
- The destWidth and destHeight parameters define the width and height of both the source and destination rectangles. This precludes any stretching.
- The source rectangle is completely within the cached bitmap dimensions. The receiving ASCE does not need to perform clipping of the source rectangle against the cached bitmap dimensions.
- The cached bitmap bits shall be interpreted using the referenced cached ColorTable.

See Table 8-71.

Table 8-71 – Memory three-way Blt order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
colorTableCacheIndex	This parameter specifies the cached ColorTable to use for this memory three-way Blt. See clause 8.16.8 for further information on ColorTable caching. For legacy mode memory three-way Blt orders, this parameter and the bitmapCacheID parameter (see below) are both present where the appropriate bit flag is set in the encodingFlags parameter in the primary order header (Note).
bitmapCacheID	This parameter, when used in conjunction with the bitmapCacheIndex parameter (see below), specifies the cached bitmap to use for this memory three-way Blt. See clause 8.16.7 for further information on bitmap caching. For legacy mode memory three-way Blt orders, this parameter and the colorTableCacheIndex parameter (see above) are both present where the appropriate bit flag is set in the encodingFlags parameter in the primary order header (Note).
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this memory three-way Blt.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this memory three-way Blt.
destWidth (coordinate)	This parameter is the width in pixels of the destination rectangle for this memory three-way Blt.
destHeight (coordinate)	This parameter is the height in pixels of the destination rectangle for this memory three-way Blt.

Table 8-71 – Memory three-way Blt order

Parameter	Description
ROP3	This parameter is the three-way ROP to use for this memory three-way Blt. For memory Blt orders, the three-way ROP shall reference a destination, source and pattern See clause 8.16.20 for further information on three-way ROPs.
sourceX (coordinate)	This parameter is the source X coordinate in the referenced cached bitmap for this memory three-way Blt.
sourceY (coordinate)	This parameter is the source Y coordinate in the referenced cached bitmap for this memory three-way Blt.
backgroundColor	This parameter is the background colour to use for this memory three-way Blt.
foregroundColor	This parameter is the foreground colour to use for this memory three-way Blt.
brush (group)	This parameter is the brush to use for this memory three-way Blt. See clause 8.16.22 for further information on brushes.
bitmapCacheIndex	This parameter, when used in conjunction with the bitmapCacheID parameter (see above), specifies the cached bitmap to use for this memory three-way Blt. See clause 8.16.7 for further information on bitmap caching.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.
NOTE – For legacy mode, memory three-way Blt orders, the colorTableCacheIndex and bitmapCacheID parameters are counted as one parameter in the primary order header encodingFlags. See clause 8.16.3.3 for further information.	

8.16.11 Text

An ASCE shall only send a text order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE), where the text order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.TextLevel capability is non-zero), the particular font has been matched and the codepoints fall within the font's specified codepoint range.

The text order allows an ASCE to specify the text start Y position in terms of either the first character's baseline or the first character cell's top. It is recommended that ASCEs use baseline positioning wherever the capabilities allow, as it may be difficult on certain terminal types to accurately position scalable font text based solely on the first character cell (left, top) position.

An ASCE shall only send text orders using baseline positioning where the negotiated capabilities indicate that baseline text start positioning is allowed (i.e., in legacy mode, the negotiated Order.textFlags capability baseline start bit flag is set; in base mode, the negotiated Order.baselineStartFlag capability is TRUE). Where that is the case, it shall set the order textFlags parameter baseline start bit flag, and set the order startX and startY as the position of the first character cell's (left, baseline) pixel. Otherwise, where the negotiated capabilities indicate that baseline text start positioning is not allowed, it shall clear the order textFlags parameter baseline start bit flag and set startX and startY as the position of the first character cell's (left, top) pixel.

Certain terminal types allow applications to dynamically specify text attributes to fonts at drawing time, in addition to those attributes that are inherent to the font. For example, if a local terminal supports a Courier bold font, then it may allow its applications to draw using that font in association with an italic attribute to achieve an emulated "Courier bold italic" (which may be different in appearance to the actual Courier bold italic font that corresponds to the available Courier bold font). Where the local terminal supports dynamic text attributes in this manner, a sending ASCE may use the textFlags parameter italic, underline and StrikeOut bit flags and the fontWeight parameter (alone or in combination) to indicate that receiving ASCEs should apply the corresponding attributes to the drawn text.

An ASCE receiving a text order draws the character codepoints on the virtual desktop, subject to any bounds clipping, as follows:

- The characters are drawn in the foregroundColor. If the backMixMode is opaque, the character cell backgrounds are drawn in the backgroundColor.
- Where the order's textFlags parameter baseline start bit flag is set, startX and startY are the position of the first character cell's (left, baseline) pixel. Otherwise, startX and startY are the position of the first character cell's (left, top) pixel.
- The characters are positioned with any extraSpacing applied to all characters and any break spacing (equal to totalBreakSpacing divided by breakCount) applied to the break character.
- The characters are drawn in the local font that corresponds to the sending ASCE's FontID, subject to any additional attributes indicated by the textFlags and/or fontWeight parameters. See clause 8.8.2 for further information on fonts.
- If the local font is scalable, the characters are drawn at the supplied fontWidth and fontHeight. Otherwise, they are drawn at the width and height inherent in the font.
- The characters are drawn using the supplied fontWeight and as per the italic, underline and StrikeOut bit flags supplied in the textFlags parameter.

See Table 8-72.

Table 8-72 – Text order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
backMixMode	This parameter is the background mix mode to use for this text order. See clause 8.16.24 for further information on background mix modes.
startX (coordinate)	This parameter is the start X virtual desktop coordinate for this text order.
startY (coordinate)	This parameter is the start Y virtual desktop coordinate for this text order. Where the baseline start bit flag is set in the textFlags parameter, this corresponds to the baseline pixel of the first character cell. Where the baseline start bit flag is not set in the textFlags parameter, this corresponds to the top pixel of the first character cell.
backgroundColor	This parameter is the background colour to use for this text order.
foregroundColor	This parameter is the foreground colour to use for this text order.
extraSpacing	This parameter specifies the additional spacing in pixels to be applied to individual characters for this text order. A value of zero indicates that no additional spacing is to be applied.
totalBreakSpacing	This parameter specifies the total of additional spacing in pixels to be applied to break characters for this text order. A value of zero indicates that no break spacing is to be applied.

Table 8-72 – Text order

Parameter	Description
breakCount	This parameter specifies the number of break characters in this text order. A value of zero indicates either that no break spacing is to be applied or there are no break characters in this order.
fontHeight	For scalable fonts, this parameter is the height in pixels of the font to use for this text order. See clause 8.8 for further information.
fontWidth	For scalable fonts, this parameter is the average character width in pixels of the font to use for this text order. See clause 8.8 for further information.
fontWeight	This parameter indicates the weight of the font to use for this text order. The allowable values are 0..1000, which are interpreted as: light ≤ 400< normal ≤ 700< bold. See clause 8.8 for further information.
textFlags	This parameter is a set of bit flags indicating additional text characteristics for this text order. Defined bit flag values are as follows: • Italic. • Underline. • StrikeOut. • Baseline start.
fontID	This parameter is the sending ASCE's FontID, determined through font negotiation, to use for this text order. See clause 8.8 for further information on font negotiation and font mapping.
codePointList	This parameter is a list of codepoints to use for this text order. The codepoints shall be in the AS protocol code page. See clause 8.8.1 for further information on codepoints and code pages.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.12 Extended text

An ASCE shall only send an extended text order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE), where the extended text order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.ExtendedTextLevel capability is non-zero), the particular font has been matched and the codepoints fall within the font's specified codepoint range.

- An ASCE shall not send an extended text order without codepoints to draw opaque rectangles alone. It should instead use the opaque rectangle order (see clause 8.16.15).
- An ASCE may send an extended text order with DeltaX positioning adjustments for fonts that approximately match with respect to X positioning.

The extended text order allows an ASCE to specify the text startY position in terms of either the first character's baseline or the first character cell's top. It is recommended that ASCEs use baseline positioning wherever the capabilities and the local terminal characteristics allow, as it may be difficult on certain terminal types to accurately position scalable font text based solely on the first character cell (left, top) position.

An ASCE shall only send extended text orders using baseline position where the negotiated capabilities indicate that baseline text start positioning is allowed (i.e., in legacy mode, the negotiated Order.textFlags capability baseline start bit flag is set; in base mode, the negotiated Order.baselineStartFlag capability is TRUE). Where that is the case, it shall set the order textFlags1 parameter baseline start bit flag, and set the order startX and startY as the position of the first character cell's (left, baseline) pixel. Otherwise, where the negotiated capabilities indicate that baseline text start positioning is not allowed, it shall clear the order textFlags1 parameter baseline start bit flag and set startX and startY as the position of the first character cell's (left, top) pixel.

Certain terminal types allow applications to dynamically specify text attributes to fonts at drawing time, in addition to those attributes that are inherent to the font. For example, if a local terminal supports a Courier bold font, then it may allow its applications to draw using that font in association with an italic attribute to achieve an emulated "Courier bold italic" (which may be different in appearance to the actual Courier bold italic corresponding to the available Courier bold font). Where the local terminal supports dynamic text attributes in this manner, a sending ASCE may use the textFlags1 parameter italic, underline and StrikeOut bit flags and the fontWeight parameter (alone or in combination) to indicate that receiving ASCEs should apply the corresponding attributes to the drawn text.

An ASCE receiving an extended text order draws the text string codepoints on the virtual desktop, subject to any bounds clipping, as follows:

- The characters are drawn in the foregroundColor. If the order's textFlags2 opaque rectangle bit flag is set, the order clip rectangle is drawn in the backgroundColor. If the order's textFlags2 opaque rectangle bit flag is not set, and the backMixMode is opaque, the character cell backgrounds are drawn in the backgroundColor.
- Where the order's textFlags1 parameter baseline start bit flag is set, startX and startY are the position of the first character cell's (left, baseline) pixel. Otherwise, startX and startY are the position of the first character cell's (left, top) pixel.
- If the order's textFlag2 parameter DeltaXPresent bit flag is set, the characters are positioned using the delta X values provided in the deltaXList parameter. If not, the characters are positioned with any extraSpacing applied to all characters and any break spacing (equal to breakTotalSpacing divided by breakCount) applied to the break character.
- The characters are drawn in the local font that corresponds to the sending ASCE's FontID, subject to any additional attributes indicated by the textFlags1 and/or fontWeight parameters. See clause 8.8.2 for further information on fonts.
- If the local font is scalable, the characters are drawn at the supplied fontWidth and fontHeight. Otherwise they are drawn at the width and height inherent in the font.
- The characters are drawn using the supplied fontWeight and as per the italic, underline and StrikeOut bit flags supplied in the order's textFlags1 parameter.
- If the order's textFlag2 clip to rectangle bit flag is set, the characters are clipped to the order clip rectangle.

See Table 8-73.

Table 8-73 – Extended text order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
backMixMode	This parameter is the background mix mode to use for this extended text order. See clause 8.16.24 for further information on background mix modes.

Table 8-73 – Extended text order

Parameter	Description
startX (coordinate)	This parameter is the start X virtual desktop coordinate for this extended text order.
startY (coordinate)	This parameter is the start Y virtual desktop coordinate for this extended text order. Where the baseline start bit flag is set in the textFlags1 parameter, this corresponds to the baseline pixel of the first character cell. Where the baseline start bit flag is not set in the textFlags1 parameter, this corresponds to the top pixel of the first character cell.
backgroundColor	This parameter is the background colour to use for this extended text order.
foregroundColor	This parameter is the foreground colour to use for this extended text order.
extraSpacing	This parameter specifies the additional spacing in pixels to be applied to individual characters for this extended text order. A value of zero indicates that no additional spacing is to be applied.
totalBreakSpacing	This parameter specifies the total of additional spacing in pixels to be applied to break characters for this extended text order. A value of zero indicates that no break spacing is to be applied.
breakCount	This parameter specifies the number of break characters in this extended text order. A value of zero indicates either that no break spacing is to be applied or there are no break characters in this order.
fontHeight	For scalable fonts, this parameter is the height in pixels of the font to use for this extended text order. See clause 8.8 for further information.
fontWidth	For scalable fonts, this parameter is the average character width in pixels of the font to use for this extended text order. See clause 8.8 for further information.
fontWeight	This parameter indicates the weight of the font to use for this extended text order. The allowable values are 0..1000, which are interpreted as: light ≤ 400< normal ≤ 700< bold. See clause 8.8 for further information.
textFlags1	This parameter is a set of bit flags indicating the characteristics of the font to use for this extended text order. Defined bit flag values are as follows. • Italic. • Underline. • StrikeOut. • Baseline start.
fontID	This parameter is the sending ASCE's FontID, determined through font negotiation, to use for this extended text order. See clause 8.8 for further information on font negotiation and font mapping.
textFlags2	This parameter is a set of bit flags specifying additional options for this extended text order. Defined bit flag values are as follows. • Opaque rectangle. • Clip to rectangle. • DeltaXPresent.
clipLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the clip rectangle for this extended text order. Where either of the textFlags2 opaque rectangle or clip to rectangle bit flags are not set, this parameter shall be zero.

Table 8-73 – Extended text order

Parameter	Description
clipTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the clip rectangle for this extended text order. Where either of the textFlags2 opaque rectangle or clip to rectangle bit flags are not set, this parameter shall be zero.
clipRight (coordinate)	This parameter is the right X virtual desktop coordinate of the clip rectangle for this extended text order. Where either of the textFlags2 opaque rectangle or clip to rectangle bit flags are not set, this parameter shall be zero.
clipBottom (coordinate)	This parameter is the bottom Y virtual desktop coordinate of the clip rectangle for this extended text order. Where either of the textFlags2 opaque rectangle or clip to rectangle bit flags are not set, this parameter shall be zero.
codePointList	This parameter is a list of codepoints to use for this extended text order. The codepoints shall be in the AS protocol code page. See clause 8.8.1 for further information on codepoints and code pages.
deltaXList (coordinate)	This parameter is a list of delta X virtual desktop coordinates to use for this extended text order. This parameter is only present where the DeltaXPresent bit flag is set in the textFlags2 parameter.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.13 Frame

An ASCE shall only send the frame order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and the frame order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.FrameLevel capability is non-zero).

An ASCE receiving a frame order performs the ROP3 parameter raster operation on the brush and the frame border on the virtual desktop, subject to any bounds clipping.

- The outer frame rectangle is defined by the destLeft, destTop, destRight and destBottom parameters.
- The inner frame rectangle is obtained by adding or subtracting the destWidth and destHeight parameters to/from the horizontal and vertical edges respectively of the outer frame rectangle.

See Table 8-74.

Table 8-74 – Frame order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this frame order.

Table 8-74 – Frame order

Parameter	Description
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this frame order.
destRight (coordinate)	This parameter is the right X virtual desktop coordinate of the destination rectangle for this frame order.
destBottom (coordinate)	This parameter is the bottom Y virtual desktop coordinate of the destination rectangle for this frame order.
destWidth (coordinate)	This parameter is the width in pixels of the rectangle vertical borders for this frame order.
destHeight (coordinate)	This parameter is the height in pixels of the horizontal rectangle borders for this frame order.
ROP3	This parameter is the three-way ROP to use for this frame order. For frame orders, the three-way ROP shall not reference a source. See clause 8.16.20 for further information on three-way ROPs.
backgroundColor	This parameter is the background colour to use for this frame order.
foregroundColor	This parameter is the foreground colour to use for this frame order.
brush (group)	This parameter is the brush to use for this frame order. See clause 8.16.22 for further information on brushes.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.14 Rectangle

An ASCE shall only send the rectangle order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and the rectangle order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.RectangleLevel capability is non-zero).

An ASCE receiving a rectangle order performs two associated operations. It uses the ROP2 parameter raster operation, the brush and (depending on the brush type) the background mix mode to draw the interior of the rectangle on the virtual desktop, subject to any bounds clipping. It also uses the ROP2 parameter raster operation, the pen and the background mix mode to draw a bounding line around the rectangle on the virtual desktop, subject to any bounds clipping.

- The interior of the rectangle is defined by destLeft + 1, destTop + 1, destRight –1 and destBottom –1.
- The line is defined by the point sequence (destLeft, destTop), (destRight, destTop), (destRight, destBottom), (destLeft, destBottom), (destLeft, destTop).

See Table 8-75.

Table 8-75 – Rectangle order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
backMixMode	This parameter is the background mix mode to use for this rectangle order. See clause 8.16.24 for further information on background mix modes.
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this rectangle order.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this rectangle order.
destRight (coordinate)	This parameter is the right X virtual desktop coordinate of the destination rectangle for this rectangle order.
destBottom (coordinate)	This parameter is the bottom Y virtual desktop coordinate of the destination rectangle for this rectangle order.
backgroundColor	This parameter is the background colour to use for this rectangle order.
foregroundColor	This parameter is the foreground colour to use for this rectangle order.
brush (group)	This parameter is the brush to use for this rectangle order. See clause 8.16.22 for further information on brushes.
ROP2	This parameter is the two-way ROP to use for this rectangle order. See clause 8.16.21 for further information on two-way ROPs.
pen (group)	This parameter is the pen to use for this rectangle order. See clause 8.16.23 for further information on pens.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.15 Opaque rectangle

An ASCE shall only send the opaque rectangle order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and the opaque rectangle order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.OpaqueRectangleLevel capability is non-zero).

An ASCE receiving an opaque rectangle order fills the destination rectangle on the virtual desktop with the supplied colour, subject to any bounds clipping. See Table 8-76.

Table 8-76 – Opaque rectangle order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
destLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the destination rectangle for this opaque rectangle order.
destTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the destination rectangle for this opaque rectangle order.
destWidth (coordinate)	This parameter is the width in pixels of the rectangle for this opaque rectangle order.

Table 8-76 – Opaque rectangle order

Parameter	Description
destHeight (coordinate)	This parameter is the height in pixels of the rectangle for this opaque rectangle order.
color	This parameter is the colour to use for this opaque rectangle order.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.16 Line

An ASCE shall only send the line order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated receiveOrdersFlag capability is TRUE) and the line order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport entry is non-zero; in base mode, the negotiated Order.LineLevel capability is non-zero).

An ASCE receiving a line order uses the ROP2 parameter raster operation, the pen and the background mix mode to draw a line from startX, startY to endX, endY on the virtual desktop, subject to any bounds clipping. Note that the line foreground colour is specified in the pen. See Table 8-77.

Table 8-77 – Line order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
backMixMode	This parameter is the background mix mode to use for this line order. See clause 8.16.24 for further information on background mix modes.
startX (coordinate)	This parameter is the start X virtual desktop coordinate for this line order.
startY (coordinate)	This parameter is the start Y virtual desktop coordinate for this line order.
endX (coordinate)	This parameter is the end X virtual desktop coordinate for this line order.
endY (coordinate)	This parameter is the end Y virtual desktop coordinate for this line order.
backgroundColor	This parameter is the background colour to use for this line order.
ROP2	This parameter is the two-way ROP to use for this line order. See clause 8.16.21 for further information on two-way ROPs.
pen (group)	This parameter is the pen to use for this line order. See clause 8.16.23 for further information on pens.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.17 Desktop save

A terminal window manager (if present) may provide local operations to save and restore local desktop areas, either automatically or under local application control, to enhance performance for typical application scenarios. This feature is referred to as "save/restore" or as "save-under". For example:

- a user drops down a menu that obscures a portion of the application's own window (area A);
- the user then selects a menu item that opens a dialogue (and dismisses the menu) that obscures a different portion of the application window (area B);
- the user then selects a dialogue button that opens a further overlapping dialogue that obscures some of the first dialogue and/or a different portion of the application window (area C);
- the user then dismisses both dialogues.

Where the window manager provides a save/restore mechanism, area A is saved when the menu drops down and restored when it is dismissed. Similarly, area B is saved when the first dialogue window is created, area C is saved when the second dialogue window is created, and then areas C and B are restored as their respective dialogue windows are destroyed. Note that multiple areas may be saved concurrently, but that the sequencing of saves and restores is such that they behave as a last-in, first-out stack.

Using save/restores is often significantly faster than forcing a window manager and/or application repaint of the obscured areas. The desktop save order allows an ASCE to propagate local save/restores to other ASCEs and often provides significant AS protocol performance benefits.

The desktop save mechanism requires that each supporting ASCE allocates a desktop cache for each other hosting ASCE to contain saved areas. Where a save occurs that affects hosted application(s) on a particular ASCE, that ASCE sends a desktop save order to all other ASCEs, specifying the area to be saved, whereupon receiving ASCEs save the corresponding area from the virtual desktop to the desktop cache. When the corresponding restore occurs, the ASCE sends a desktop save order to all other ASCEs, specifying the area to be restored, whereupon the receiving ASCEs restore the corresponding area from the desktop cache to the virtual desktop.

The desktop cache is logically organized as a set of tiles, the size of which are a specific X and Y granularity. The preferred granularity for a particular ASCE depends on the relative efficiency of save and restores for different size tiles between the virtual desktop and the desktop cache and the degree of memory wastage as the tile size increases. However, while a particular ASCE may advertise its preferred granularities, those values are negotiated, and the ASCE has to be prepared to receive desktop save orders constructed with respect to larger (and possibly less efficient) values. See Appendix I for details of informative values for desktop cache sizing and granularity.

A desktop save order specifies an action (which is either save or restore), the area to save/restore (as a rectangle) and a pixel offset into the desktop cache. The sending ASCE calculates the pixel offset based on the previous usage of the other ASCEs' desktop cache and the negotiated X and Y granularity as follows:

- for each desktop save order: at initialization, set the accumulated pixel offset to zero;
- calculate the pixels required for this save/restore as follows: the pixel offset for the order is the accumulated pixel offset:

  area width in pixels = ((area width in pixels + (negotiated X granularity – 1)) div negotiated X granularity) * negotiated X granularity;

  area height in pixels = ((area height in pixels + (negotiated Y granularity – 1)) div negotiated Y granularity) * negotiated Y granularity;

  pixels required for this area = area width in pixels * area height in pixels;
- if this is a save operation, add the pixels required for this area to the accumulated pixel offset;
- if this is a restore operation, subtract the pixels required for this area to the accumulated pixel offset.

An ASCE shall only send the desktop save order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., in legacy mode, the negotiated Order.orderFlags capability does not have the cannot receive orders bit flag set; in base mode, the negotiated Order.receiveOrdersFlag capability is TRUE) and the desktop save order is supported (i.e., in legacy mode, the corresponding negotiated Order.orderSupport capability entry is non-zero; in base mode, the negotiated Order.DesktopSaveLevel capability is non-zero).

Where the sending ASCE detects that another ASCE's desktop cache is full, then it shall not send further desktop save orders until it can restore areas already in the desktop cache. This may require that it takes locally-dependent action on its terminal to emulate the save operation and/or force a local repaint on the restore. See Table 8-78.

Table 8-78 – Desktop save order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
saveOffset	This parameter is the pixel offset within the receiving ASCE's desktop cache to use for this desktop save order action.
desktopLeft (coordinate)	This parameter is the left X virtual desktop coordinate of the desktop rectangle for this desktop save order action.
desktopTop (coordinate)	This parameter is the top Y virtual desktop coordinate of the desktop rectangle for this desktop save order action.
desktopRight (coordinate)	This parameter is the right X virtual desktop coordinate of the desktop rectangle for this desktop save order action.
desktopBottom (coordinate)	This parameter is the bottom Y virtual desktop coordinate of the destination rectangle for this desktop save order action.
action	This parameter identifies this desktop save order action. The allowable values are DesktopSave and DesktopRestore.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.18 Desktop origin

Where an ASCE's local desktop is smaller than the virtual desktop, then an ASCE may provide locally-defined mechanisms to scroll the local desktop over the virtual desktop, so that an end user

can view and control shadow windows hosted on ASCEs with larger desktops. This mechanism is referred to as desktop scrolling.

Where an ASCE implements desktop scrolling, the local desktop behaves as a viewport onto the virtual desktop and the local desktop origin may be offset from the virtual desktop origin.

Where an ASCE is hosting windows and its local desktop origin is not coincident with the virtual desktop origin, it shall send a desktop origin order to inform other active ASCEs of its desktop origin in virtual desktop coordinates – that is, with respect to the virtual desktop origin. This mechanism ensures that an ASCE always knows the desktop origin that applies for each incoming ASPDU and for each order within an UpdatePDU (orders) from each ASCE.

An ASCE shall ensure that it sends a desktop origin order before sending any ASPDUs containing virtual desktop coordinates that are calculated with respect to that desktop origin.

An ASCE shall ensure that where it sends a desktop origin order, then any virtual desktop coordinates sent preceding that desktop origin order are calculated with reference to the old desktop origin and any virtual desktop coordinates sent following that desktop origin order are calculated with reference to the new desktop origin. This condition may require that an ASCE flush orders and/or other ASPDUs before or after sending a desktop origin order.

An ASCE may use desktop origin orders from a particular ASCE to implement a desktop scrolling mechanism for shadow windows hosted by that ASCE, by subtracting the sending ASCE's desktop origin from virtual desktop coordinates in incoming ASPDUs.

In the legacy mode of the AS protocol, an ASCE shall only send the desktop origin order where the negotiated Order.orderSupport capability entry for the screen Blt order is non-zero. In the base mode of the AS protocol, an ASCE shall only send the desktop origin order where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., the negotiated Order.receiveOrdersFlag capability is TRUE) and the desktop origin order is supported (i.e., the negotiated Order.DesktopOriginLevel capability is non-zero). See Table 8-79.

Table 8-79 – Desktop origin order

Parameter	Description
Primary order header	The primary order header is described in clause 8.16.1.
desktopOriginX (coordinate)	This parameter is the X virtual desktop coordinate of the sending ASCE's local desktop origin.
desktopOriginY (coordinate)	This parameter is the Y virtual desktop coordinate of the sending ASCE's local desktop origin.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.19 Colour space

In the base mode of the AS protocol, an ASCE may use the colour space order to specify the colour space that is in effect for subsequent orders and provide optional colour accuracy information. An ASCE may also use the colour space order to restore the default colour space – RGB with no colour accuracy information. The colour space order is not supported in the legacy mode of the AS protocol.

An ASCE shall only send the colour space order in the base mode of the AS protocol, where the negotiated capabilities indicate that all other ASCEs can receive orders (i.e., the negotiated

Order.receiveOrdersFlag capability is TRUE) and the colour space order is supported (i.e., the negotiated Order.ColorSpaceLevel capability is non-zero).

An ASCE receiving a colour space order should make best efforts to interpret subsequent order colour information with respect to the colour space specified in the colour space order. See Table 8-80.

Table 8-80 – Colour space order

Parameter	Description
colorSpace	This parameter is the colour space and optional colour accuracy information to use when interpreting subsequent orders.
nonStandardParameters	This parameter is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.16.20 Three-way ROPs

Three-way ROPs are ternary raster-operation codes used by the following orders.

- Destination Blt: See clause 8.16.4.
- Pattern Blt: See clause 8.16.5.
- Screen Blt: See clause 8.16.6.
- Memory Blt: See clause 8.16.9.
- Memory three-way Blt: See clause 8.16.10.
- Frame: See clause 8.16.13.

Ternary raster operation codes define the combination of the bits in a source bitmap and a brush (also known as a pattern) with the bits in a destination bitmap. They are bit-wise operations acting on individual bits without colour interpretation, where the bits may be part of palette indices or direct colour values.

The following are the three operands used in these operations:

- D Destination bitmap.
- P Selected brush (also called a pattern or stipple).
- S Source bitmap.

The following are the Boolean operators used in these operations:

- a Bit-wise AND.
- n Bit-wise NOT (inverse).
- o Bit-wise OR.
- x Bit-wise exclusive OR (XOR).

All combinations of Boolean operations can be presented in reverse polish notation. For example, the following operation replaces the bits in the destination bitmap with a bit-wise OR of the bits in the source and brush:

PSo

Again, the following operation bit-wise ORs the bits in the source and brush with the bits in the destination bitmap:

DPSoo

Each raster operation code is a single octet value that represents the result of the boolean operation on predefined brush, source and destination values. For example, the raster operation codes for the PSo and DPSoo operations are shown in the following list:

Source values			Generated values	
P	**S**	**D**	**PSo**	**DPSoo**
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1
			⇒0xFC	⇒0xFE

In this case, PSo has the raster operation code 0xFC (reading the generated bit values as an octet with the least significant bit at the top); DPSoo has the raster operation code 0xFE.

The raster operation codes can be interpreted directly to determine if the ternary raster operation references a source, pattern or destination operand, as follows:

- ternary raster operation does not reference a pattern: bits 0..3 = bits 4..7;
- ternary raster operation does not reference a source: bits 0, 1, 4, 5 = bits 2, 3, 6, 7;
- ternary raster operation does not reference a destination: bits 0, 2, 4, 6 = bits 1, 3, 5, 7.

The full list of 256 ternary raster operation codes is shown in Table 8-81, with the reverse polish definition for each code[4].

Note that, because raster operations act directly on bit values, without colour interpretation, the results of raster operations may not deliver predictable colour effects when applied on different ASCEs. This may be particularly noticeable for raster operations that reference a destination operand, such as destination invert (i.e., Dn or ROP code 0x55 – see below) which is often used to implement "rubber-band" effects). If a hosted application uses a raster operation of this type, where the hosting ASCE is running on a terminal using a different local colour depth and/or colour model and where the local operation is sent as an order, then the resultant colour(s) may be very different between the hosting ASCE and other ASCEs.

Consider two ASCEs participating in application sharing within a conference, where ASCE A is hosting applications. ASCE A's local terminal uses a 8 bits-per-pixel palettized RGB colour model whereas ASCE B's local terminal uses a 4 bits-per-pixel palettized RGB colour model. While the difference in colour depths will affect the negotiated Bitmap.sendingBitsPerPixel and therefore the colour depth of palette and ColorTable information sent from ASCE A to ASCE B, at some point ASCE B has to map the protocol colour information, orders and bitmap data onto its local terminal display, which is 4 bits-per-pixel. The following example omits intermediate mapping stages (e.g., from the protocol information to any receiving cache to the local terminal display) and concentrates on the net colour mapping as it affects destination raster operations.

---

[4] There are alternative reverse polish spellings of the same raster operation code, so although a particular spelling may not be in the table, an equivalent form is present. For example, DSa is equivalent to SDa.

- A colour-aware application running on ASCE A's local terminal adds colours to a portion of the local terminal palette, such that its 8 bits-per-pixel palette contains (among others) the following entries:
  - palette index 0x63 ⇒ RGB <204,0,102> ("violet red");
  - palette index 0x9C ⇒ RGB <204,204,204> ("light grey").
- ASCE B's local terminal does not permit application palette management. Its 4 bits-per-pixel palette contains (among others) the following entries:
  - palette index 0x1 ⇒ RGB <128,0,0> ("dark red");
  - palette index 0xE ⇒ RGB <0,255,255> ("cyan").
- An application on ASCE's local terminal fills an area of a hosted window to RGB <204,0,102> using a local graphics operation. The local terminal graphics system generates an area of screen pixels containing palette index 0x63 ("violet red").
- ASCE A sends a pattern Blt order for that area, using the raster operation 0xCC (S) and foreground colour RGB <204,0,102>. This assumes a one-to-one mapping between local colour information and the protocol (which is not necessarily always the case).
- ASCE B draws the pattern Blt order to the local terminal display, generating an area of screen pixels containing palette index 0x1 ("dark red"). The colour mapping may be an explicit mapping by ASCE B or may rely on a local colour mapping provided by the local terminal.
- The application on ASCE A's local terminal then inverts the filled area using a local graphics operation. This results in the area of screen pixels being inverted to 0x9C ("light grey").
- ASCE A sends a destination Blt order for the area, using the raster operation 0x55 (Dn).
- ASCE B draws the destination Blt order to the local terminal display. This results in the area of screen pixels being inverted to 0xE ("cyan").

This (admittedly somewhat artificial) example illustrates that, while the initial colour mapping from "violet red" to "dark red" is reasonable, within the constraints of the local terminal's colour models, "cyan" is not a good mapping for "light grey" – the effect of the destination raster operation is to arbitrarily move colour values (by changing the colour indices directly) without reference to colour mappings. In practice, the extent of destination raster operation artefacts depends on the particular local terminal palettes in use – where both local terminals are using linear palettes (of the kind that are typically used as default local terminal palettes) artefacts are relatively benign, but where one or more local terminals or their local applications are using non-linear palettes or explicitly manipulating the local terminal palette contents, destination raster operation artefacts may be significant.

While the above discussion indicates that application sharing introduces additional colour fidelity issues through the use of orders containing raster operations referencing a destination operand, the extent of those issues is terminal- and application-dependent. However, sending such operations as orders rather than bitmap data typically delivers significantly better performance. Therefore, ASCEs are recommended to provide local terminal mechanisms whereby users can enable or disable the sending of orders containing raster operations referencing a destination operand, so that users can choose between colour fidelity and/or performance on a per-terminal and/or application basis. See Table 8-81.

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
00	0
01	DPSoon
02	DPSona
03	PSon
04	SDPona
05	DPon
06	PDSxnon
07	PDSaon
08	SDPnaa
09	PDSxon
0A	DPna
0B	PSDnaon
0C	SPna
0D	PDSnaon
0E	PDSonon
0F	Pn
10	PDSona
11	DSon
12	SDPxnon
13	SDPaon
14	DPSxnon
15	DPSaon
16	PSDPSanaxx
17	SSPxDSxaxn
18	SPxPDxa
19	SDPSanaxn
1A	PDSPaox
1B	SDPSxaxn
1C	PSDPaox
1D	DSPDxaxn
1E	PDSox
1F	PDSoan
20	DPSnaa
21	SDPxon
22	DSna
23	SPDnaon
24	SPxDSxa

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
25	PDSPanaxn
26	SDPSaox
27	SDPSxnox
28	DPSxa
29	PSDPSaoxxn
2A	DPSana
2B	SSPxPDxaxn
2C	SPDSoax
2D	PSDnox
2E	PSDPxox
2F	PSDnoan
30	PSna
31	SDPnaon
32	SDPSoox
33	Sn
34	SPDSaox
35	SPDSxnox
36	SDPox
37	SDPoan
38	PSDPoax
39	SPDnox
3A	SPDSxox
3B	SPDnoan
3C	PSx
3D	SPDSonox
3E	SPDSnaox
3F	PSan
40	PSDnaa
41	DPSxon
42	SDxPDxa
43	SPDSanaxn
44	SDna
45	DPSnaon
46	DSPDaox
47	PSDPxaxn
48	SDPxa
49	PDSPDaoxxn

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
4A	DPSDoax
4B	PDSnox
4C	SDPana
4D	SSPxDSxoxn
4E	PDSPxox
4F	PDSnoan
50	PDna
51	DSPnaon
52	DPSDaox
53	SPDSxaxn
54	DPSonon
55	Dn
56	DPSox
57	DPSoan
58	PDSPoax
59	DPSnox
5A	DPx
5B	DPSDonox
5C	DPSDxox
5D	DPSnoan
5E	DPSDnaox
5F	DPan
60	PDSxa
61	DSPDSaoxxn
62	DSPDoax
63	SDPnox
64	SDPSoax
65	DSPnox
66	DSx
67	SDPSonox
68	DSPDSonoxxn
69	PDSxxn
6A	DPSax
6B	PSDPSoaxxn
6C	SDPax
6D	PDSPDoaxxn
6E	SDPSnoax

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
6F	PDSxnan
70	PDSana
71	SSDxPDxaxn
72	SDPSxox
73	SDPnoan
74	DSPDxox
75	DSPnoan
76	SDPSnaox
77	DSan
78	PDSax
79	DSPDSoaxxn
7A	DPSDnoax
7B	SDPxnan
7C	SPDSnoax
7D	DPSxnan
7E	SPxDSxo
7F	DPSaan
80	DPSaa
81	SPxDSxon
82	DPSxna
83	SPDSnoaxn
84	SDPxna
85	PDSPnoaxn
86	DSPDSoaxx
87	PDSaxn
88	DSa
89	SDPSnaoxn
8A	DSPnoa
8B	DSPDxoxn
8C	SDPnoa
8D	SDPSxoxn
8E	SSDxPDxax
8F	PDSanan
90	PDSxna
91	SDPSnoaxn
92	DPSDPoaxx
93	SPDaxn

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
94	PSDPSoaxx
95	DPSaxn
96	DPSxx
97	PSDPSonoxx
98	SDPSonoxn
99	DSxn
9A	DPSnax
9B	SDPSoaxn
9C	SPDnax
9D	DSPDoaxn
9E	DSPDSaoxx
9F	PDSxan
A0	DPa
A1	PDSPnaoxn
A2	DPSnoa
A3	DPSDxoxn
A4	PDSPonoxn
A5	PDxn
A6	DSPnax
A7	PDSPoaxn
A8	DPSoa
A9	DPSoxn
AA	D
AB	DPSono
AC	SPDSxax
AD	DPSDaoxn
AE	DSPnao
AF	DPno
B0	PDSnoa
B1	PDSPxoxn
B2	SSPxDSxox
B3	SDPanan
B4	PSDnax
B5	DPSDoaxn
B6	DPSDPaoxx
B7	SDPxan
B8	PSDPxax

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
B9	DSPDaoxn
BA	DPSnao
BB	DSno
BC	SPDSanax
BD	SDxPDxan
BE	DPSxo
BF	DPSano
C0	PSa
C1	SPDSnaoxn
C2	SPDSonoxn
C3	PSxn
C4	SPDnoa
C5	SPDSxoxn
C6	SDPnax
C7	PSDPoaxn
C8	SDPoa
C9	SPDoxn
CA	DPSDxax
CB	SPDSaoxn
CC	S
CD	SDPono
CE	SDPnao
CF	SPno
D0	PSDnoa
D1	PSDPxoxn
D2	PDSnax
D3	SPDSoaxn
D4	SSPxPDxax
D5	DPSanan
D6	PSDPSaoxx
D7	DPSxan
D8	PDSPxax
D9	SDPSaoxn
DA	DPSDanax
DB	SPxDSxan
DC	SPDnao
DD	SDno

Table 8-81 – Three-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
DE	SDPxo
DF	SDPano
E0	PDSoa
E1	PDSoxn
E2	DSPDxax
E3	PSDPaoxn
E4	SDPSxax
E5	PDSPaoxn
E6	SDPSanax
E7	SPxPDxan
E8	SSPxDSxax
E9	DSPDSanaxxn
EA	DPSao
EB	DPSxno
EC	SDPao
ED	SDPxno
EE	DSo
EF	SDPnoo
F0	P
F1	PDSono
F2	PDSnao
F3	PSno
F4	PSDnao
F5	PDno
F6	PDSxo
F7	PDSano
F8	PDSao
F9	PDSxno
FA	Dpo
FB	DPSnoo
FC	Pso
FD	PSDnoo
FE	DPSoo
FF	1

8.16.21 Two-way ROPs

Two-way ROPs are binary raster-operation codes used by the following orders:

- Rectangle See clause 8.16.14.
- Line See clause 8.16.16.

Binary raster operation codes define the combination of the bits in a pen or brush with the bits in a destination bitmap. They are bit-wise operations acting on individual bits without colour interpretation, where the bits may be part of palette indices or direct colour values.

The following are the two operands used in these operations:

- D Destination bitmap.
- P Pen (or brush).

The following are the Boolean operators used in these operations:

- a Bit-wise AND.
- n Bit-wise NOT (inverse).
- o Bit-wise OR.
- x Bit-wise exclusive OR (XOR).

All combinations of Boolean operations can be presented in reverse polish notation. For example, the following operation replaces bits in the destination bitmap with a bit-wise OR of bits in the destination bitmap and the pen/brush.

DPo

Each binary raster operation code is a single octet value consisting of the result of the Boolean operation on predefined pen/brush and destination values, which is then incremented by one. For example, the raster operation codes for the DPo and DPan operations are shown in the following list:

Source values		**Generated values**	
P	**D**	**DPo**	**DPan**
0	0	0	1
1	1	1	1
1	0	1	1
1	1	1	0
		⇒0x0E + 1	⇒0x07 + 1

In this case, DPo has the raster operation code 0x0F (reading the generated bit values as an octet with the least significant bit at the top – and then incrementing); DPan has the raster operation code 0x08.

Note that, because raster operations act directly on bit values, without colour interpretation, the results of raster operations may not deliver predictable colour effects when applied on different ASCEs. This may be particularly noticeable for raster operations that reference a destination operand, such as destination invert (i.e., Dn or ROP code 0x6 – see below) which is often used to implement "rubber-band" effects). If a hosted application uses a raster operation of this type, where the hosting ASCE is running on a terminal using a different local colour depth and/or colour model and where the local operation is sent as an order, then the resultant colour(s) may be very different between the hosting ASCE and other ASCEs. See clause 8.16.20 for further discussion.

The full list of 16 binary raster operation codes is shown in Table 8-82, with the reverse polish definition for each code[5].

Table 8-82 – Two-way ROPs definition

ROP code (Hexadecimal)	Reverse polish
01	0
02	DPon
03	DPna
04	Pn
05	PDna
06	Dn
07	DPx
08	DPan
09	DPa
0A	DPxn
0B	D
0C	DPno
0D	P
0E	PDno
0F	DPo
10	1

8.16.22 Brushes

Brushes (also known as patterns) are used to paint interior areas. They are used by the following orders:

- Pattern Blt: See clause 8.16.5.
- Memory three-way Blt: See clause 8.16.10.
- Frame: See clause 8.16.13.
- Rectangle: See clause 8.16.14.

The brush parameters are subject to order parameter encoding. See clause 8.16.3.3 for further information. See Table 8-83.

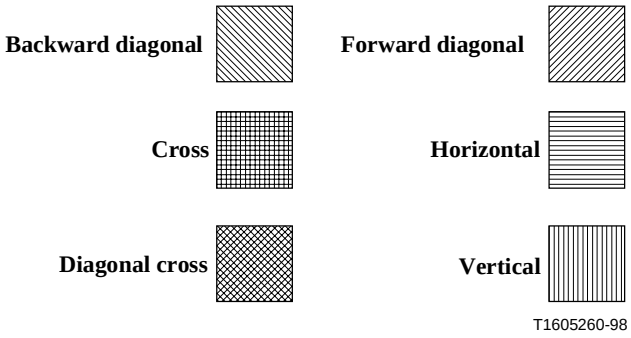
[5] There are alternative reverse polish spellings of the same raster operation code, so although a particular spelling may not be in the table, an equivalent form is present. For example, DPx is equivalent to PDx.

Table 8-83 – Brush definition

Parameter	Description
originX	This parameter is ignored where the brush type is solid or null. Where the brush type is hatch or pattern, this parameter is the X coordinate of the brush origin.
originY	This parameter is ignored where the brush type is solid or null. Where the brush type is hatch or pattern, this parameter is the Y coordinate of the brush origin.
Style	This parameter indicates the style to use for this brush. Allowable values are as follows: • Solid brush — The interior area is painted using the order foreground colour and the ROP2 or ROP3 in the order. • Null brush — The brush does not contribute to the painting of the interior area. • Hatched brush — The interior area is painted using the brush hatch style in the order foreground and background colours, with respect to the brush origin and using the ROP2 or ROP3 in the order. • Pattern brush — The interior area is painted using the brush pattern in the order foreground and background colours, with respect to the brush origin and using the ROP2 or ROP3 in the order.
Hatch	Where the style parameter indicates a hatched brush, this parameter indicates the hatch style to use for this brush. Allowable values are as follows: • Horizontal hatch • Vertical hatch • Forward diagonal hatch • Backward diagonal hatch • Cross hatch • Diagonal cross hatch Where the style parameter indicates a pattern brush, this parameter is the first octet of the pattern.
pattern	Where the style parameter indicates a pattern brush, this parameter contains octets two through eight of the pattern.
nonStandardBrushParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard brush parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

Table 8-84 shows how the various brush hatch styles appear when used to paint the interior of a rectangle.

Table 8-84 – Brush hatch styles



8.16.22.1 Brush patterns

A brush pattern consists of a bicolour 8 pixel by 8 pixel bitmap, where one bits are drawn in the background colour and zero bits are drawn in the foreground colour.

The pattern bits are sent as 8 octets of uncompressed bitmap data, arranged so that pixel x, y is in octet y + 1, bit position (x mod 8). The first octet is sent in the hatch parameter and octets two through eight are sent in the pattern parameter.

8.16.22.2 Brush origin

When a hatch or pattern brush is used to paint an interior area, the brush is replicated to fill the area, with respect to the brush origin. The brush origin defines the position of the brush (top, left) pixel on the virtual desktop from which replication should start. The brush replication process can be summarized as follows:

- position the brush (top, left) pixel at the brush origin on the virtual desktop;
- copy the brush horizontally and vertically throughout the virtual desktop, clipping to the interior area.

8.16.23 Pens

Pens are used to paint lines. They are used by the following orders:

- Rectangle: See clause 8.16.14.
- Line: See clause 8.16.16.

The AS protocol supports the following pen types:

- Solid.
- Dashed.
- Dotted.
- Dash-dot.
- Dash-dot-dot.
- Null.

The pen parameters are subject to order parameter encoding. See clause 8.16.3.3 for further information. See Table 8-85.

Table 8-85 – Pen definition

Parameter	Description
style	This parameter indicates the style to use for this pen. Allowable values are as follows: • Solid pen. • Dashed pen. • Dotted pen. • Dash-dot pen. • Dash-dot-dot pen. • Null pen.
width	This parameter indicates the width to use for the pen. The allowable value is 1.
color	This parameter is the colour to use for the pen.
nonStandardPenParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard pen parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

For a solid pen, all pixels along the path of the line are drawn using the designated pen colour and the ROP2 in the order.

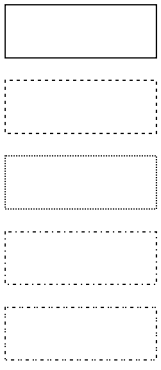
For dashed, dotted, dash-dot or dash-dot-dot pens, the line is drawn using a foreground/background pixel sequence determined by the pen style, where:

- foreground pixels are drawn using the pen foreground colour and the ROP2 in the order;
- background pixels are not drawn if the background mix is transparent (see clause 8.16.24);
- background pixels are drawn using the order background colour and ROP2 if the background mix is opaque (see clause 8.16.24).

For a null pen, no line pixels are drawn.

Table 8-86 shows how the various pen styles appear when used to draw a rectangle border.

Table 8-86 – Pen styles



T1605270-98

8.16.24 Background mix

A background mix is used in the following orders:

- Text: See clause 8.16.11.
- Extended text: See clause 8.16.12.
- Rectangle: See clause 8.16.14.
- Line: See clause 8.16.16.

For the text and extended text orders, the background mix determines whether background pixels are drawn or not for the text character cells. Where the background mix is transparent, character cell background pixels are not drawn. Where the background mix is opaque (and for the extended text order, an opaque rectangle is not required), the character cell background pixels are drawn in the order's background colour.

For the rectangle and line orders, the background mix determines whether pen off pixels are drawn for dashed, dotted, dash-dot and dash-dot-dot pen styles. Where the background mix is transparent, pen off pixels are not drawn. Where the background mix is opaque, the pen off pixels are drawn using the order's background colour and the ROP2 parameter. See clause 8.16.23 for further information on pens and 8.16.21 for further information on binary raster operation codes.

For the rectangle order, the background mix also determines whether interior background pixels are drawn when a hatched brush is used. For this case, where the background mix is transparent, background pixels are not drawn, but, where the background mix is opaque, background pixels are drawn using the order's background colour and the ROP2 parameter. The background mix does not affect interior background pixels for other brush styles – for solid brushes, background pixels are not applicable; for pattern brushes, background pixels are always drawn.

The AS protocol supports the following background mix types. See Table 8-87.

Table 8-87 – Background mix types

Background mix type	Definition
Opaque	Background pixels are drawn.
Transparent	Background pixels are not drawn.

8.17 Bitmap updates

An ASCE sends bitmap updates to all ASCEs within the conference by sending an UpdatePDU containing bitmap data in the manner indicated in Table 6-3. The content of the UpdatePDU containing bitmap data is shown in Table 8-88.

An ASCE shall only send uncompressed bitmap data where the negotiated Bitmap.sendingBitsPerPixel capability value is one, four or eight. An ASCE shall only send compressed bitmap data where the negotiated Bitmap.sendingBitsPerPixel capability is four or eight, and the negotiated Bitmap.bitmapCompressionFlag capability is TRUE. Compressed bitmap data shall not be sent where the negotiated Bitmap.sendingBitsPerPixel capability is one.

On receipt of an UpdatePDU containing bitmap data, the receiving ASCE (after any required decompression) performs a copy from (top, left) of the bitmap to (top, left) of the destination rectangle on the virtual desktop for the destination width and height. Where the bitmap width and/or height are larger than the destination width and/or height, the receiving ASCE shall clip the bitmap to the destination rectangle. The ASCE shall interpret bitmap pixel values using the last received UpdatePDU (palette) ASPDU. See clause 8.15 for further information on palettes.

Table 8-88 – UpdatePDU (bitmap)

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
destLeft	This parameter is the left X virtual desktop coordinate of the destination rectangle for the bitmap data.
destTop	This parameter is the top Y virtual desktop coordinate of the destination rectangle for the bitmap data.
destRight	This parameter is the right X virtual desktop coordinate of the destination rectangle for the bitmap data.
destBottom	This parameter is the bottom Y virtual desktop coordinate of the destination rectangle for the bitmap data.
width	This parameter is the width in pixels of the bitmap data. The bitmap width shall be larger than or equal to the width of the destination rectangle (see above).
height	This parameter is the height in pixels of the bitmap data. The bitmap height shall be larger than or equal to the height of the destination rectangle (see above).
bitsPerPixel	This parameter is the bits-per-pixel. This parameter shall be the same as the negotiated Bitmap.sendingBitsPerPixel value. See clause 8.2.4 for further information.
compressedFlag	This parameter indicates whether the bitmap data (see below) is compressed or not. A value of TRUE indicates that the bitmap data is compressed and a value of FALSE that it is not.
bitmapData	This parameter is the bitmap data. The bitmap may be uncompressed (see clause 8.17.1) or compressed (see clause 8.17.2).
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.17.1 Uncompressed bitmap data

Uncompressed bitmap data represents the bitmap as a series of rows, where row zero starts at the highest pixel Y coordinate. That is, row 0 starts at (bottom, left).

Within a row, pixel values are packed into octets, starting from the left-most pixel:

- For 1 bits-per-pixel bitmap data, each octet contains eight pixels, with the left-most pixel in the most significant bit.
- For 4 bits-per-pixel bitmap data, each octet contains two pixels, with the left-most pixel in the most significant four bits.
- For 8 bits-per-pixel bitmap data, each octet contains one pixel value.

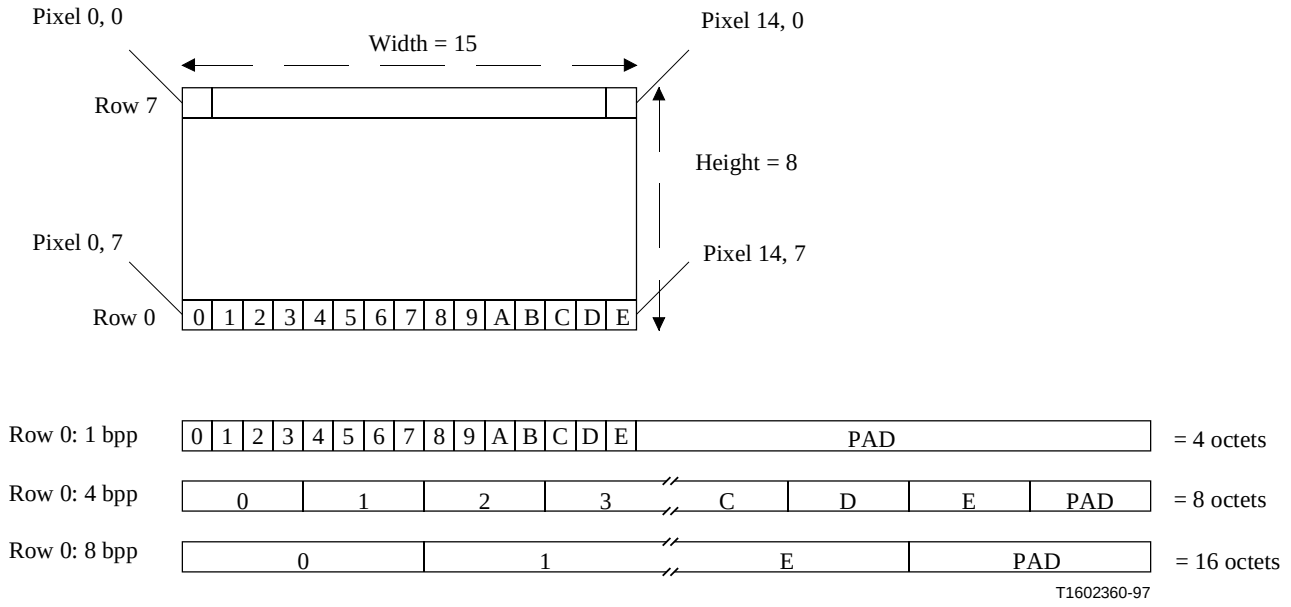
Each row of bitmap data is padded to a four-octet boundary. This means that the size of a row is:

((((bitmap-width-in-pixels * bit-per-pixel) + 31) div 32) * 4) octets.

The total size of the bitmap data is:

(height-of-bitmap * octets-per-row) octets.

The following diagram illustrates the layout of pixels in uncompressed bitmap data for row 0 of a 15×8 bitmap, at 1, 4 and 8 bits-per-pixel, respectively.



8.17.2 Compressed bitmap data

For compressed bitmap data, the actual compressed bitmap information is preceded by the compressed bitmap header described in Table 8-89.

Table 8-89 – Compressed bitmap header

Parameter	Description
mainBodySize	This parameter is the length in octets of the compressed bitmap data (after compression).
rowSize	This parameter is the length in octets of a single row of the uncompressed bitmap data.
uncompressedSize	This parameter is the length in octets of the uncompressed bitmap data.
compressedBitmapData	This parameter is the compressed bitmap data.

AS protocol bitmap compression is oriented to the compression of window manager data, where runs may occur in both X and Y. The AS protocol represents a compressed bitmap as a sequence of compression codes, where a particular bitmap may be represented by different valid code sequences.

Compression codes are of two types:

- Difference codes are two-dimensional runs where a series of pixel values are determined by the compression code and the corresponding pixel values in previous rows. The Fill, Mix, FillOrMix, SetMix_Mix, SetMix_FillOrMix, FillOrMix_1, and FillOrMix_2 compression codes (see Table 8-90) are difference codes, which are defined in terms of one or more fill and/or mix operation(s).
  - The fill operation sets the current pixel in the current row to the corresponding pixel in the previous row. This is a straightforward copy operation.
  - The mix operation sets the current pixel in the current row to the exclusive OR (XOR) of the current mix value and the corresponding pixel in the previous row. The mix value defaults to 0xFF (which is set at the start of a compression/decompression pass for each new bitmap) and may be updated by the SetMix_Mix and SetMix_FillOrMix codes.

Because different codes reference the previous bitmap row, they have defined special case operations for processing the first row (see the individual code operation definitions in Table 8-90).

- Colour codes are linear runs where a series of pixel values are determined by reference to the compression code alone. The Color, Copy, CopyPacked, Bicolour, Black and White compression codes (see Table 8-90) are colour codes.

A compression code consists of a code identifier, followed by an optional length field, followed by optional associated data dependent on the code. A particular compression code may have multiple encodings, dependent on the size of the length field. See Table 8-90 for further information on encodings with respect to length.

The length field may be further encoded dependent on the code. See Table 8-91 for further information on length encodings. Where present, the length (after any required decoding) may specify the number of octets of associated data and the required repeat.

A receiving ASCE interprets a sequence of bitmap compression codes contained within a UpdatePDU (bitmap) to generate a decompressed bitmap (in the format described in clause 8.17.1), as follows:

1) Set the initial pixel position to (bottom, left); set the initial mix value to 0xFF.
2) Decode the next code and calculate its length value (n).
3) Perform the defined code operation for the defined repeat (wrapping to the next row as necessary).
4) Update the pixel position (wrapping to the next row as necessary).
5) Repeat steps 2 through 4 until all codes are processed.

For example, if the current pixel position is pixel 60 in row 6 and the next code is fill, with encoding 3b<0×0>,5b<len = 8>, the receiving ASCE sets the values of pixel 60..67 to the values in pixels 60..67 in row 5 and updates the pixel position to 68.

Where the original bitmap (prior to compression) contains padding (see clause 8.17.1), the sending ASCE shall treat the padding as pixel data for compression purposes. This allows the receiving ASCE to treat the compression code sequence as referring to a linear series of pixels (independent of padding considerations):

- Table 8-90 summarizes the encodings, type of length encoding, operation and repeat for each defined code.
- Table 8-91 summarizes the encoding for each length type.
- Table 8-92 summarizes the notation used in Tables 8-90 and 8-91.

The compressed bitmap data is packed into the compressedBitmapData parameter by filling in successive bits into each octet beginning with the most significant bit of each field and filling towards the least significant bit. Where fields are less than an octet, successive fields are filled into the next free most significant bit. Where a single field occupies two octets, octets are filled in increasing significance, with the highest-order, or most significant, octet placed in the second octet[6].

[6] While two octet fields have octets filled in increasing significance, the constituent bits within each octet continue to be filled beginning with the most significant bit of each field and filling towards the least significant bit.

Table 8-90 – Bitmap compression codes

Code	Encoding	Length type	Operation	Repeat
Fill 9 (Note 1)	3b<0x0>,5b<len> 8b<0x00>,8b<len> 8b<0xF0>,16b<len>	A	row 1: $pixel^{i} \Leftarrow 0$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1}$	i = 0..n–1
Mix	3b<0x1>,5b<len> 8b<0x20>,8b<len> 8b<0xF1>,16b<len>	A	row 1: $pixel^{i} \Leftarrow mix$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1} \wedge mix$	i = 0..n–1
FillOrMix	3b<0x2>,5b<len>,(n+7)/8<mask> 8b<0x40>,8b<len>,(n+7)/8<mask> 8b<0xF2>,16b<len>,(n+7)/8<mask>	A8	row 1: $pixel^{i} \Leftarrow 0$ or $pixel^{i} \Leftarrow mix$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1}$ or $pixel^{i} \Leftarrow pixel^{i,r-1} \wedge mix$	i = 0..n–1 for (n+7)/8 masks
Color	3b<0x3>,5b<len>,<color> 8b<0x60>,8b<len>,<color> 8b<0xF3>,16b<len>,<color>	A	$pixel^{i} \Leftarrow color$	i = 0..n–1
Copy	3b<0x4>,5b<len>,n<color> 8b<0x80>,8b<len>,n<color> 8b<0xF4>,16b<len>,n<color>	A	$pixel^{i} \Leftarrow color^{i}$	i = 0..n–1
CopyPacked (Note 2)	3b<0x5>,5b<len>,n/2<packed_color> 8b<0xA0>,8b<len>,n/2<packed_color> 8b<0xF5>,16b<len>,n/2<packed_color>	A	$pixel^{i*2} \Leftarrow packed\_color^{i}$(high), $pixel^{(i*2)+1} \Leftarrow packed\_color^{i}$(low)	i = 0..(n/2)–1
SetMix_Mix	4b<0xC>,4b<len>,<color> 8b<0xC0>,8b<len>,<color> 8b<0xF6>,16b<len>,<color>	B	mix ⇐ color, row 1: $pixel^{i} \Leftarrow mix$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1} \wedge mix$	i = 0..n–1
SetMix_FillOrMix	4b<0xD>,4b<len>,<color>,(n+7)/8<mask> 8b<0xD0>,8b<len>,<color>,(n+7)/8<mask> 8b<0xF7>,16b<len>,<color>,(n+7)/8<mask>	B8	mix ⇐ color, row 1: $pixel^{i} \Leftarrow 0$ or $pixel^{i} \Leftarrow mix$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1}$ or $pixel^{i} \Leftarrow pixel^{i,r-1} \wedge mix$	i = 0..n–1 for (n+7)/8 masks
Bicolor	4b<0xE>,4b<len>,<color1>,<color2> 8b<0xE0>,8b<len>,<color1>,<color2> 8b<0xF8>,16b<len>,<color1>,<color2>	B	$pixel^{i*2} \Leftarrow color1$, $pixel^{(i*2)+1} \Leftarrow color2$	i = 0..n–1
FillOrMix_1	8b<0xF9>{,<mask=0x03>}	C	row 1: $pixel^{i} \Leftarrow 0$ or $pixel^{i} \Leftarrow mix$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1}$ or $pixel^{i} \Leftarrow pixel^{i,r-1} \wedge mix$	i = 0..7 for 1 mask
FillOrMix_2	8b<0xFA>{,<mask=0x05>}	C	row 1: $pixel^{i} \Leftarrow 0$ or $pixel^{i} \Leftarrow mix$ row 2..H: $pixel^{i} \Leftarrow pixel^{i,r-1}$ or $pixel^{i} \Leftarrow pixel^{i,r-1} \wedge mix$	i = 0..7 for 1 mask
White	8b<0xFD>	C	pixel ⇐ 0xF (4bpp)/0xFF (8bpp)	1
Black	8b<0xFE>	C	pixel ⇐ 0	1

NOTE 1 – Where this fill immediately follows another fill (i.e., there are consecutive adjoining fill codes), the receiving ASCE shall insert a single mix code of encoding 3b<0x1>,5b<len=1>.

NOTE 2 – An ASCE shall only send compressed bitmap data containing the CopyPacked code where the negotiated Bitmap.sendingBitsPerPixel capability value is four and it is sending four bits-per-pixel compressed bitmap data. See clause 8.2.4 for further information.

Table 8-91 – Bitmap compression codes: Length encoding

Length type	Range		Calculation
A	5b:	1..31	n = len
	8b:	0..255	n = len + 32
	16b:	288..65535	n = len
B	4b:	1..15	n = len
	8b:	0..255	n = len + 16
	16b:	272..65535	n = len
A8	5b:	1..31	n = len * 8
	8b:	0..254	n = len + 1
	16b:	256..65535	n = len
B8	4b:	1..31	n = len * 8
	8b:	0..254	n = len + 1
	16b:	256..65535	n = len
C	n/a		n = 1

Table 8-92 – Bitmap compression codes: Legend

Notation	Description
nb<f>	The field f is n bits in size. For example, mix may be encoded with an initial 3-bit field of value 0x1.
<len>	The field contains a length of the type specified in the length column. The extracted length value (calculated depending on encoding as per Table 8-91) is referred to as n in the remainder of the encoding and is typically used to specify the number of following octets of mask, color or packed_color fields and to generate the repeat.
n<f>	There are n occurrences of field type f.

<mask>	The field contains a mask. A mask is an octet encoding 8 pixels as bits. A one-bit value indicates the mix value should be used (see below) and a zero bit value indicates the fill value should be used. The pixel-to-bit encoding is as follows:
	pixels (highest pixel address = 7) ⇒ bits (most significant bit = 7)
	`0 1 2 3 4 5 6 7` `7 6 5 4 3 2 1 0`
	`a b c d e f g h` `h f d b g e c a`
	For example, a mask of 0xAA (i.e., bits 10101010) expands to a series of pixels of fill, fill, mix, mix, fill, fill, mix, mix.

<color>	The field contains an octet of colour information. For four bits-per-pixel bitmap data, the most significant four bits should be ignored. The bitmap compression encoding does not place any interpretation on colour values.
<packed_color>	The field contains a packed octet of colour value. The packed colour value is only allowable for four bits-per-pixel bitmap data, where it consists of two four-bit colour values, where the first colour is in the most significant four bits and the second colour is in the least significant four bits. The bitmap compression encoding does not place any interpretation on colour values.
<f1>,<f2>	Field f1 precedes field f2. For example, fill may be encoded by a 3-bit field of 0x0, followed by a 5-bit field giving the length.

Table 8-92 – Bitmap compression codes: Legend

Notation	Description
{,<f=value>}	Field f with value is implied and is not included by the sending ASCE. For example, code FillOrMix_1 has an initial octet of 0xF9 which implies mask value 0x03.
p ⇐ e	Pixel p is assigned the expression e. For example, the operation for black is pixel ⇐ black, which replaces the current pixel with the black colour value (see below).
p ⇐ e1∧e2	Pixel p is assigned the bit-wise exclusive OR (XOR) of the two expressions.
op1, op2	Perform operation op1 then operation op2.
op1 or op2	Perform operation op1 for corresponding zero mask bits and operation op2 for corresponding one mask bits, using the mask pixel to bit encoding (see above).
row 1: op1 or row 2..H:op2	Perform operation op1 for pixels in the first row of the destination bitmap and operation op2 for pixels in all other rows.
$Pixel^{i}$	The current pixel in the current row of the destination bitmap. Where the code has a repeat, i takes the values indicated by the repeat column.
$Pixel^{i,r-1}$	The corresponding pixel in the previous row of the destination bitmap. Where the code defines a repeat, i takes the values indicated by the repeat column. For example, if the current pixel is pixel 35 in row 6, $pixel^{i,r-1}$ is pixel 35 in row 5.
Mix	The current mix value. The mix value is reset to 0xFF at the start of decompression of a new UpdatePDU (bitmap) or cache bitmap order (see clause 8.16.7) and is thereafter set by the SetMix_Mix and/or SetMix_FillOrMix codes.

8.18 Input

Input handling is tightly integrated with conducted mode (if in effect within the conference) and with the conference control policy. Depending on conductorship and/or control state, the ASCE may or may not have the right to provide input to hosted or shadow windows within the conference. See clause 8.19 for further information on conducted mode and clauses 8.12 and 8.13 for further information on control mechanisms.

An ASCE may provide input to peer ASCEs within the conference by sending an InputPDU containing input events in the manner indicated in Table 6-3. The content of the InputPDU is shown in Table 8-93. Input events may be of one of the following three types:

- Pointing device event: See clause 8.18.1.
- Keyboard event: See clause 8.18.2.
- Input synchronization event: See clause 8.18.6.

Table 8-93 – InputPDU

Parameter	Description
ShareData header	The ShareData header is described in clause 8.3.
eventList	This parameter is a list of input events (see below).
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.18.1 Pointing device events

Depending on conductorship and control state, an ASCE may queue a pointing device event within an InputPDU when a local pointing device event occurs. A receiving ASCE shall process pointing device events with respect to the current local control state and policy. The pointing device event is described in Table 8-94.

Pointing device events model a three-button pointing device, where:

- Button 1 is the logical left button (commonly used for selection).
- Button 2 is the logical right button.
- Button 3 is the logical middle button.

Certain terminals allow for re-assignment of the local logical to physical pointing device button mapping, under end-user or programmatic control, such that (for example) the logical left (or selection) button is swapped to the physical right button. Where such a facility is provided, the ASCE shall ensure that pointing device events are sent in terms of the logical button assignments.

Pointing device events contain the virtual desktop coordinate of the point where the event occurred. The sending ASCE is responsible for constraining the event to the virtual desktop. Receiving ASCEs are responsible for constraining any local pointing device events generated from received pointing device events to the local desktop.

Table 8-94 – Pointing device event

Parameter	Description
eventTime	This parameter is the local ASCE time in milliseconds when this event occurred.
pointingDeviceFlags	This parameter is a set of bit flags identifying and qualifying the pointing device event. Defined bit flag values are as follows: • Move. • Button 1. • Button 2. • Button 3. • Down. See below for further information on the interpretation of pointing device event flags.
pointingDeviceX	This parameter is the X virtual desktop coordinate of the pointing device event.
pointingDeviceY	This parameter is the Y virtual desktop coordinate of the pointing device event.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

Table 8-95 describes the usage of the pointing device flags for a three-button pointing device. The sending ASCE is responsible for setting the pointing device flags in pointing device events. Receiving ASCEs are responsible for interpreting the pointing device flags with respect to the local terminal characteristics.

The AS protocol does not support pointing device double-click events. Where a double-click occurs on the local terminal, an ASCE shall send the corresponding pointing device event sequence button down, button up, button down, button up (e.g., for a button 1 double-click, the ASCE shall send button 1 down, button 1 up, button 1 down, button 1 up), with the relevant local event times. A receiving ASCE is responsible for processing that pointing device event sequence such that, where the event times fall within the local terminal double-click interval, it is interpreted as a double-click. This means that in a multipoint conference, where an ASCE is displaying shadow windows corresponding to hosted windows on multiple hosting ASCEs and the local ASCE is cooperating and in control, then the double-click time will depend on which local and/or shadow window has the input focus.

Table 8-95 – Three button pointing device event flags

Pointing device event	Valid combinations of pointingDeviceFlags
Move	Move
Left button down	Button 1 + down
Left button up	Button 1
Right button down	Button 2 + down
Right button up	Button 2
Middle button down	Button 3 + down
Middle button up	Button 3

8.18.2 Keyboard events

Depending on conductorship and control state, an ASCE may queue a keyboard event within an InputPDU when a local keyboard event occurs. A receiving ASCE shall process keyboard events with respect to the current local control state and policy.

Keyboard events can be represented in the AS protocol as either code points or virtual keys (see below). The keyboard event is described in Table 8-96.

Table 8-96 – Keyboard event

Parameter	Description
eventTime	This parameter is the local ASCE time in milliseconds when this event occurred.
keyboardflags	This parameter is a set of bit flags identifying and qualifying the keyboard event. Defined bit flag values are as follows: • Right. • Quiet. • Down. • Release. The right and quiet bit flags are only valid for virtual key keyboard events. The down and release bit flags are valid for both codepoint and virtual key keyboard events. See below for further information on the interpretation of keyboard event flags.
keyCode	For codepoint keyboard events, this parameter is an AS protocol code page codepoint (see clause 8.8.1). For virtual key keyboard events, this parameter is a virtual keycode (see clause 8.18.3 below).

Table 8-96 – Keyboard event

Parameter	Description
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

A sending ASCE uses combinations of the down and release bit flags in the keyboardFlags parameter to indicate the following keyboard actions.

- Key release: down and release bit flags are set.
- Key press: Neither down nor release bit flags are set.
- Key repeat: Down bit flag is set.

A receiving ASCE:

- processes received virtual key keyboard events with respect to the previously established keyboard modifier and toggle state, but does not interpret nor translate the virtual keycode value;
- processes received code point keyboard events by generating the appropriate series of local actions required to produce the codepoint on the local terminal.

Where a sending ASCE can use either a codepoint or a virtual key for a particular key (e.g., AS protocol codepoint 0x41 and virtual keycode 0x41 are both uppercase A), it should send the codepoint rather than the virtual keycode. This is preferred, as AS protocol code page codepoints are independent of keyboard modifier and/or toggle state, whereas virtual keycodes are not. For example, on US PC 101 and Mac keyboards the @ (Unicode 0x26, AMPERSAND) character is shift-2 whereas on UK PC 101 keyboards it is shift-‘ (shift-quote) – and shift-2 is " (Unicode 0x22, QUOTATION MARK). Therefore, if an ASCE executing on a US PC was in a conference where peer ASCEs had a mix of UK and US keyboards and sent virtual keycodes, then typing shift-2 locally – which would produce the sequence of virtual keycode keyboard events VK_SHIFT-down, VK_2-down, VK_2-up, VK_SHIFT-up – might variously produce @ and/or " on other terminals. In contrast, if the ASCE sent codepoints, which would produce the sequence of codepoint, keyboard events 0x0026-down, 0x0026-up, each peer ASCE could take appropriate local action to produce the @.

8.18.3 Virtual keycodes

Virtual keycodes provide a hardware- and language-independent method of identifying keyboard keys. Each virtual keycode represents a unique keyboard key and also identifies the purpose of that key. Table 8-97 defines the virtual keycodes supported by the AS protocol. ASCEs shall only send InputVirtualKey keyboard events in InputPDUs using defined virtual keycodes.

On certain terminals, keys may be duplicated (for example, shift is typically duplicated to the left and right), but are only allocated one virtual keycode (VK_SHIFT = 0x10). Some applications are sensitive to whether end-users have pressed the left or right variant of such keys. Therefore, where an ASCE can locally distinguish between the left and right variants it should:

- set or clear the right bit flag in the keyboardFlags parameter when sending such virtual keycode events;
- check the right bit flag in the keyboardFlags parameter when receiving virtual keycode events and take the appropriate local action to generate the corresponding local indication.

Where an ASCE cannot distinguish between left and right variants, then it should:

- never set the right bit flag in the keyboardFlags parameter when sending virtual keycode events;
- never check the right bit flag in the keyboardFlags parameter when receiving virtual keycode events – and therefore handle events that have the right bit flag set as if the event referenced the left variant.

Table 8-97 – Virtual keycodes

Name	Value	Comment
VK_CANCEL	0x03	Break
VK_BACK	0x08	
VK_TAB	0x09	
VK_CLEAR	0x0C	
VK_RETURN	0x0D	
VK_SHIFT	0x10	
VK_CONTROL	0x11	
VK_ALT	0x12	Also known as menu
VK_PAUSE	0x13	
VK_CAPITAL	0x14	
VK_ESCAPE	0x1B	
VK_SPACE	0x20	
VK_PRIOR	0x21	Page up
VK_NEXT	0x22	Page down
VK_END	0x23	
VK_HOME	0x24	
VK_LEFT	0x25	
VK_UP	0x26	
VK_RIGHT	0x27	
VK_DOWN	0x28	
VK_SELECT	0x29	
VK_SNAPSHOT	0x2C	Print screen
VK_INSERT	0x2D	
VK_DELETE	0x2E	
VK_HELP	0x2F	
VK_0	0x30	Numerals...
VK_1	0x31	
VK_2	0x32	
VK_3	0x33	
VK_4	0x34	
VK_5	0x35	
VK_6	0x36	
VK_7	0x37	

Table 8-97 – Virtual keycodes

Name	Value	Comment
VK_8	0x38	
VK_9	0x39	
VK_A	0x41	Alphabetics...
VK_B	0x42	
VK_C	0x43	
VK_D	0x44	
VK_E	0x45	
VK_F	0x46	
VK_G	0x47	
VK_H	0x48	
VK_I	0x49	
VK_J	0x4A	
VK_K	0x4B	
VK_L	0x4C	
VK_M	0x4D	
VK_N	0x4E	
VK_O	0x4F	
VK_P	0x50	
VK_Q	0x51	
VK_R	0x52	
VK_S	0x53	
VK_T	0x54	
VK_U	0x55	
VK_V	0x56	
VK_W	0x57	
VK_X	0x58	
VK_Y	0x59	
VK_Z	0x5A	
VK_LEFT_MENU	0x5B	Left menu key
VK_RIGHT_MENU	0x5C	Right menu key
VK_CONTEXT	0x5D	Context menu key
VK_NUMPAD0	0x60	Numeric pad keys...
VK_NUMPAD1	0x61	
VK_NUMPAD2	0x62	
VK_NUMPAD3	0x63	
VK_NUMPAD4	0x64	
VK_NUMPAD5	0x65	
VK_NUMPAD6	0x66	

Table 8-97 – Virtual keycodes

Name	Value	Comment
VK_NUMPAD7	0x67	
VK_NUMPAD8	0x68	
VK_NUMPAD9	0x69	
VK_MULTIPLY	0x6A	
VK_ADD	0x6B	
VK_SUBTRACT	0x6D	
VK_DECIMAL	0x6E	
VK_DIVIDE	0x6F	
VK_F1	0x70	Function keys...
VK_F2	0x71	
VK_F3	0x72	
VK_F4	0x73	
VK_F5	0x74	
VK_F6	0x75	
VK_F7	0x76	
VK_F8	0x77	
VK_F9	0x78	
VK_F10	0x79	
VK_F11	0x7A	
VK_F12	0x7B	
VK_F13	0x7C	
VK_F14	0x7D	
VK_F15	0x7E	
VK_F16	0x7F	
VK_F17	0x80	
VK_F18	0x81	
VK_F19	0x82	
VK_F20	0x83	
VK_F21	0x84	
VK_F22	0x85	
VK_F23	0x86	
VK_F24	0x87	
VK_NUMLOCK	0x90	Num lock
VK_SCROLL	0x91	Scroll lock
VK_PLUS	0xBB	
VK_COMMA	0xBC	
VK_MINUS	0xBD	
VK_PERIOD	0xBE	

Table 8-97 – Virtual keycodes

Name	Value	Comment
VK_BAR	0xE2	Solid bar (non-US)
VK_ATTN	0xF6	
VK_CRSEL	0xF7	
VK_EXSEL	0xF8	
VK_EREOF	0xF9	
VK_PLAY	0xFA	
VK_ZOOM	0xFB	
VK_PA1	0xFD	

8.18.4 Keyboard state

During an AS session, an ASCE need only send keyboard events when providing input to a remotely hosted application. It need not send keyboard events for keys typed before it became active, or when the conference is in conducted mode and it does not have conductorship privileges, or it is not in control, or it is detached.

A sending ASCE shall ensure that the portions of the local input stream that are sent in keyboard events in InputPDUs are internally consistent. For example, suppose that the local end-user had CapsLock on before the ASCE became active in the AS session, and then takes control of a remotely hosted application and starts typing. The ASCE must precede sent keyboard events with additional keyboard events containing CapsLock down/up events, and rely on peer ASCEs to prepare their local keyboard state so that the CapsLock down/up takes effect (which may be a null operation if CapsLock was on remotely as well) to ensure that the input is capitalized on the remote system.

The general requirement is that the sending ASCE inserts keyboard events as required into the outgoing input stream to synchronize the remote keyboard to the local keyboard state.

During an AS session, an ASCE may receive input from multiple peer ASCEs. For example, where an ASCE is hosting an application in a multipoint conference, several peer ASCEs may provide input in turn, as the remote end-users exchange control and provide input to the hosted application. While an ASCE can rely on the sending ASCEs inserting any required state (see above), it is responsible for undoing any local keyboard state established on behalf of other peer ASCEs before processing input from a new ASCE. For example, if ASCE A is hosting the application, ASCE B is in control and has CapsLock on and control switches to ASCE C who already has NumLock on, then when ASCE A switches input from ASCE B to ASCE C, it can rely on ASCE C prefacing its input stream with a NumLock-down, NumLock-up, but is itself responsible for undoing the CapsLock state established on behalf of ASCE B. When ASCE A subsequently switches input from ASCE C back to ASCE B, it is then responsible for re-establishing the CapsLock on state on behalf of ASCE B.

The general requirement is that a receiving ASCE is responsible for taking any local action required to synchronize the local keyboard state to that of a new input stream.

8.18.5 Quiet keys

Many terminals use special key sequences to effect actions that should not be reflected at peer ASCEs. For example, ALT-TAB is commonly used to cycle the focus around window manager windows.

Such sequences should only have local effect, but a sending ASCE may only be able to detect them after the first event(s) in the sequence have been sent. Where this is the case, the sending ASCE should send subsequent events with the quiet bit flag set in the keyboardFlags parameter to indicate to peer ASCEs that they should undo the effect of the sequence.

For example, an ASCE may send ALT-TAB as: ALT-down, TAB-down-quiet, TAB-up-quiet, ALT-up-quiet.

Where a receiving ASCE detects that the quiet bit flag is set for a down event, that key and its associated up event may be discarded. Where the receiving ASCE only detects that the quiet bit flag is set after the down event for a key, then there may be local mechanisms whereby the effect of the key can be suppressed by inserting additional key events before or after the associated up event.

For example, an ASCE may process the sent ALT-TAB sequence above as: ALT-down, discard TAB-down-quiet, discard TAB-up-quiet, insert terminal-dependent suppression events, ALT-up.

8.18.6 Input synchronization event

When an ASCE detects that a new ASCE has become active, it shall reset its sending keyboard state and shall queue an input synchronization event for sending within the next InputPDU. Thereafter, when sending input events, it shall insert input events to bring peer ASCEs up to the same keyboard state as the local terminal.

On receipt of an input synchronization event within an InputPDU, an ASCE shall reset its receiving keyboard state. Thereafter, before receiving input from a particular peer ASCE, it can expect to receive any required keyboard state applicable to that input.

See clause 8.4 for further information on ASCE activation and clause 8.6 for further information on synchronization. The input synchronization event is described in Table 8-98.

Table 8-98 – Input synchronization event

Parameter	Description
eventTime	This parameter is the local ASCE time in milliseconds when this event occurred.
nonStandardParameters	This parameter is only allowed in the base mode of the AS protocol. It is an optional list of non-standard parameters allowed only if the corresponding non-standard capabilities are present in the negotiated capability set.

8.19 Conducted mode operation

When a conference is in conducted mode, the rights of an ASCE to host applications and to provide input events may be restricted by the conducting node.

When a conference enters conducted mode or when conductorship moves from one node to another, each ASCE receives a GCC-Conductor-Assign indication. Upon receipt of such an indication, no ASCE at a node other than the conducting node is permitted to host applications or provide input.

Receipt of GCC-Conductor-Permission-Grant indications while in conducted mode grants and denies the right for ASCEs at nodes other than the conducting node to host applications and provide input. The right is granted if the permission flag parameter is TRUE. The right is denied if the permission flag parameter is set to FALSE.

An ASCE at the conducting node of a conducted mode conference may host applications and provide input.

In non-conducted mode, all ASCEs may host applications and provide input, subject to the conference control policy (see clauses 8.12 and 8.13).

9 ASPDU definitions

The structure of ASPDUs for the legacy and base modes of the AS protocol are specified as follows using the notation ASN.1 of [ITU-T X.680].

Legacy mode ASPDUs are specified in clause 9.1 and shall be encoded for transmission by applying the encoding rules defined in clause 9.3.

Base mode ASPDUs are specified in clause 9.2 and shall be encoded for transmission by applying the BASIC ALIGNED variant of the packed encoding rules of [ITU-T X.691].

Both legacy and base mode ASPDUs shall be encoded and placed in the data field of MCS-Send-Data primitives, with the bit string generated by the encoding placed in the OCTET STRING used by MCS in the order such that for each octet, the leading bit is placed in the most significant bit position and the trailing bit is placed in the least significant bit position.

9.1 Legacy mode ASN.1 definition

```
--/////////////////////////////////////////////////////////////
--
--                Begin AS Definitions
--
--/////////////////////////////////////////////////////////////
AS-PROTOCOL-for-legacy-mode {itu-t(0) recommendation(0) t(20)
t128(128) version(0) 2 as-protocol-for-legacy-mode(2)}
DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

-- NOTE - ================================================
-- NOTE - All abstract types defined shall be exported
-- NOTE - ================================================

-- Constants
maxSourceDescriptor INTEGER ::= 48
maxTerminalDescriptor INTEGER ::= 16
maxFonts INTEGER ::= 700
maxPassword INTEGER ::= 9
maxFaceName INTEGER ::= 32
maxTitleString INTEGER ::= 50
maxInputEvents INTEGER ::= 50

-- Base Types
BitString8 ::= BIT STRING(SIZE (0..7))
BitString16 ::= BIT STRING(SIZE (0..15))
BitString32 ::= BIT STRING(SIZE (0..31))
Coordinate8 ::= INTEGER(127..128)
Coordinate16 ::= INTEGER(32767..32768)
Integer4 ::= INTEGER(0..15)
Integer8 ::= INTEGER(0..255)
Integer12 ::= INTEGER(0..4095)
Integer16 ::= INTEGER(0..65535)
Integer32 ::= INTEGER(0..4294967295)
Boolean16 ::= INTEGER {false(0), true(1)}(0..65535)
UserID ::= Integer16
ShareID ::= Integer32
WindowID ::= Integer32
T50String ::= OCTET STRING(SIZE (0..255)) -- T.50 String
ASString ::= OCTET STRING(SIZE (0..255)) -- AS Protocol CodePage String


-- Bit Flag Types
BitmapCompressionCapabilityFlags ::= BIT STRING {bitmapCompressionSupported(0)
}(SIZE (0..15))
```

```
BoundsOrderFlags ::= BIT STRING {
  absoluteLeftPresent(0), absoluteTopPresent(1), absoluteRightPresent(2),
  absoluteBottomPresent(3), deltaLeftPresent(4), deltaTopPresent(5),
  deltaRightPresent(6), deltaBottomPresent(7)}(SIZE (0..7))

ControlCapabilityFlags ::= BIT STRING {allowMediatedControl(0)}(SIZE (0..15))

ControlOrderFlags ::= BIT STRING {
  standard(0), -- Mandatory this flag is set
  secondary(1), bounds(2), typeChange(3), deltaCoordinates(4)}(SIZE (0..7))

ExtraOrderFlags ::= BIT STRING {secondary(3)}(SIZE (0..15))

ExtraTextFlags ::= BIT STRING {
  opaqueRectangle(1), clipToRectangle(2), deltaXPresent(15)}(SIZE (0..15))

FontAttributeFlags ::= BIT STRING {fixedPitch(0), fixedSize(1)}(SIZE (0..15))

KeyboardFlags ::= BIT STRING {right(0), quiet(12), down(14), release(15)
}(SIZE (0..15))

OrderCapabilityFlags ::= BIT STRING {
  negotiateOrderSupport(1), -- Mandatory this flag is set
  cannotReceiveOrders(2)}(SIZE (0..15))

PointingDeviceFlags ::= BIT STRING {
  move(11), button1(12), button2(13), button3(14), down(15)}(SIZE (0..15))

TextAttributeFlags ::= BIT STRING {
  italic(2), underline(3), strikeout(4), useBaselineStart(8)}(SIZE (0..15))

TextCapabilityFlags ::= BIT STRING {
  checkFontAspect(0), allowDeltaXSimulation(5), checkFontSignatures(7),
  useBaselineStart(9)}(SIZE (0..15))

WindowAttributeFlags ::= BIT STRING {
  minimized(0), taggable(1), hosted(2), shadow(3), local(4), topmost(5),
  windowManagerMinimized(16), windowManagerInvisible(17)}(SIZE (0..31))

-- General Types
ApplicationAction ::= INTEGER {
  notifyHostedApplications(1), unhostApplication(2)}(0..65535)

BackgroundMixMode ::= INTEGER {transparent(1), opaque(2)}(0..65535)

BitmapData ::= CHOICE {
  uncompressedBitmapData  OCTET STRING,
  compressedBitmapData    CompressedBitmapData
}

Brush ::= SEQUENCE {
  originX  Integer8 OPTIONAL,
  originY  Integer8 OPTIONAL,
  style    BrushStyle OPTIONAL,
  hatch    BrushHatch OPTIONAL,
  pattern  OCTET STRING(SIZE (7)) OPTIONAL
}

BrushHatch ::= CHOICE {style        HatchStyle,
                       patternZero  Integer8
}

BrushStyle ::= INTEGER {solid(0), null(1), hatched(2), pattern(3)}(0..255)
```

```
Color ::= SEQUENCE {red    Integer8,
                    green  Integer8,
                    blue   Integer8
}

ColorQuad ::= SEQUENCE {
  blue       Integer8,
  green      Integer8,
  red        Integer8,
  pad1octet  Integer8
}

ColorPointerAttribute ::= SEQUENCE {
  cacheIndex        Integer16,
  hotSpot           Point16,
  width             Integer16,
  height            Integer16,
  lengthANDMAsk     Integer16,
  -- length in octets of AND mask in colorPointerData
  lengthXORMAsk     Integer16,
  -- length in octets of XOR mask in colorPointerData
  colorPointerData  OCTET STRING
}

CompressedBitmapData ::= SEQUENCE {
  pad2octets        Integer16(0),
  mainBodySize      Integer16,
  rowSize           Integer16,
  uncompressedSize  Integer16,
  compressedBitmap  OCTET STRING
}

ControlAction ::= INTEGER {
  requestControl(1), detach(3), grantControl(2), cooperate(4)}(0..65535)

ControlPriority ::= INTEGER {always(1), never(2), confirm(3)}(0..65535)

Coordinate ::= CHOICE {absolute  Coordinate16,
                       delta     Coordinate8
}

DesktopSaveAction ::= INTEGER {desktopSave(0), desktopRestore(1)}(0..255)

FontAttribute ::= SEQUENCE {
  faceName      T50String(SIZE (1..maxFaceName)),
  fontFlags     FontAttributeFlags,
  averageWidth  Integer16,
  height        Integer16,
  aspectX       Integer16,
  aspectY       Integer16,
  signature1    Integer8,
  signature2    Integer8,
  signature3    Integer16,
  codePage      FontCodePage,
  ascent        Integer16
}

FontCodePage ::= INTEGER {allCodePoints(0), coreCodePoints(255)}(0..65535)

HatchStyle ::= INTEGER {
  horizontal(0), vertical(1), forward(2), backward(3), cross(4), diagonal(5)
}(0..255)

InputMessageType ::= INTEGER {
```

```
  inputSynchronize(0), inputCodePoint(1), inputVirtualKey(2),
  inputPointingDevice(32769) -- ('8001'H)--}(0..65535)

MediatedControlAction ::= INTEGER {
  takeControlRequest(1), passControlRequest(2), detachRequest(3),
  confirmTakeResponse(5), denyTakeResponse(6), confirmDetachResponse(7),
  denyDetachResponse(8), denyPassResponse(9), remoteDetachRequest(10),
  denyRemoteDetachRequest(11)}(0..65535)

MonoPointerAttribute ::= SEQUENCE {
  hotSpot            Point16,
  width              Integer16,
  height             Integer16,
  lengthPointerData  Integer16,
  -- length in octets of monoPointerData
  monoPointerData    OCTET STRING
}

OSMajorType ::= INTEGER {
  unspecified(0), windows(1), oS2(2), macintosh(3), unix(4)}(0..65535)

OSMinorType ::= INTEGER {
  unspecified(0), windows-31x(1), windows-95(2), windows-NT(3), oS2-V21(4),
  power-pc(5), macintosh(6), native-XServer(7), pseudo-XServer(8)}(0..65535)

PDUType ::= INTEGER {
  confirmActivePDU(3), dataPDU(7), deactivateAllPDU(6), deactivateOtherPDU(4),
  deactivateSelfPDU(5), demandActivePDU(1), requestActivePDU(2)}(0..15)

PDUType2 ::= INTEGER {
  application(25), control(20), font(11), input(28), mediatedControl(29),
  pointer(27), remoteShare(30), synchronize(31), update(2),
  updateCapability(32), windowActivation(23), windowList(24)}(0..255)

PDUTypeFlow ::= INTEGER {flowResponsePDU(66), flowStopPDU(67), flowTestPDU(65)
}(0..255)

Pen ::= SEQUENCE {
  style  PenStyle OPTIONAL,
  width  Integer8(1) OPTIONAL,
  color  Color OPTIONAL
}

PenStyle ::= ENUMERATED {
  solid(0), dashed(1), dotted(2), dash-dot(3), dash-dot-dot(4), null(5)
}

Point16 ::= SEQUENCE {x  Coordinate16,
                      y  Coordinate16
}

PointerMessageType ::= INTEGER {
  cachedPointer(7), colorPointer(6), monoPointer(2), pointerPosition(3),
  systemPointer(1)}(0..65535)

PrimaryOrderType ::= INTEGER {
  destinationBlt(0), patternBlt(1), screenBlt(2), memoryBlt(13),
  memoryThreeWayBlt(14), text(5), extendedText(6), frame(9), rectangle(7),
  line(8), opaqueRectangle(10), desktopSave(11), desktopOrigin(32)}(0..255)

Rectangle16 ::= SEQUENCE {
  left    Coordinate16,
  top     Coordinate16,
  right   Coordinate16,
```

```
  bottom  Coordinate16
}

RemoteShareAction ::= INTEGER {
  requestRemoteShare(1), confirmRemoteShare(2), denyRemoteShare(3)}(0..65535)

RemoteShareDenial ::= INTEGER {
  incorrectPassword(1), remoteShareNotEnabled(2),
  remoteShareInOperationIncoming(3), remoteShareInOperationOutgoing(4)
}(0..65535)

ROP2 ::= INTEGER {
  r2BLACK(1), r2DPon(2), r2DPna(3), r2Pn(4), r2PDna(5), r2Dn(6), r2DPx(7),
  r2DPan(8), r2DPa(9), r2DPxn(10), r2D(11), r2DPno(12), r2P(13), r2PDno(14),
  r2DPo(15), r2WHITE(16)}(0..255)

ROP3 ::= INTEGER {
  r3BLACK(0), -- ('00'H)
  r3DPSoon(1), -- ('01'H)
  r3DPSona(2), -- ('02'H)
  r3PSon(3), -- ('03'H)
  r3SDPona(4), -- ('04'H)
  r3DPon(5), -- ('05'H)
  r3PDSxnon(6), -- ('06'H)
  r3PDSaon(7), -- ('07'H)
  r3SDPnaa(8), -- ('08'H)
  r3PDSxon(9), -- ('09'H)
  r3DPna(10), -- ('0A'H)
  r3PSDnaon(11), -- ('0B'H)
  r3SPna(12), -- ('0C'H)
  r3PDSnaon(13), -- ('0D'H)
  r3PDSonon(14), -- ('0E'H)
  r3Pn(15), -- ('0F'H)
  r3PDSona(16), -- ('10'H)
  r3DSon(17), -- ('11'H)
  r3SDPxnon(18), -- ('12'H)
  r3SDPaon(19), -- ('13'H)
  r3DPSxnon(20), -- ('14'H)
  r3DPSaon(21), -- ('15'H)
  r3PSDPSanaxx(22), -- ('16'H)
  r3SSPxDSxaxn(23), -- ('17'H)
  r3SPxPDxa(24), -- ('18'H)
  r3SDPSanaxn(25), -- ('19'H)
  r3PDSPaox(26), -- ('1A'H)
  r3SDPSxaxn(27), -- ('1B'H)
  r3PSDPaox(28), -- ('1C'H)
  r3DSPDxaxn(29), -- ('1D'H)
  r3PDSox(30), -- ('1E'H)
  r3PDSoan(31), -- ('1F'H)
  r3DPSnaa(32), -- ('20'H)
  r3SDPxon(33), -- ('21'H)
  r3DSna(34), -- ('22'H)
  r3SPDnaon(35), -- ('23'H)
  r3SPxDSxa(36), -- ('24'H)
  r3PDSPanaxn(37), -- ('25'H)
  r3SDPSaox(38), -- ('26'H)
  r3SDPSxnox(39), -- ('27'H)
  r3DPSxa(40), -- ('28'H)
  r3PSDPSaoxxn(41), -- ('29'H)
  r3DPSana(42), -- ('2A'H)
  r3SSPxPDxaxn(43), -- ('2B'H)
  r3SPDSoax(44), -- ('2C'H)
  r3PSDnox(45), -- ('2D'H)
  r3PSDPxox(46), -- ('2E'H)
```

```
r3PSDnoan(47), -- ('2F'H)
r3PSna(48), -- ('30'H)
r3SDPnaon(49), -- ('31'H)
r3SDPSoox(50), -- ('32'H)
r3Sn(51), -- ('33'H)
r3SPDSaox(52), -- ('34'H)
r3SPDSxnox(53), -- ('35'H)
r3SDPox(54), -- ('36'H)
r3SDPoan(55), -- ('37'H)
r3PSDPoax(56), -- ('38'H)
r3SPDnox(57), -- ('39'H)
r3SPDSxox(58), -- ('3A'H)
r3SPDnoan(59), -- ('3B'H)
r3PSx(60), -- ('3C'H)
r3SPDSonox(61), -- ('3D'H)
r3SPDSnaox(62), -- ('3E'H)
r3PSan(63), -- ('3F'H)
r3PSDnaa(64), -- ('40'H)
r3DPSxon(65), -- ('41'H)
r3SDxPDxa(66), -- ('42'H)
r3SPDSanaxn(67), -- ('43'H)
r3SDna(68), -- ('44'H)
r3DPSnaon(69), -- ('45'H)
r3DSPDaox(70), -- ('46'H)
r3PSDPxaxn(71), -- ('47'H)
r3SDPxa(72), -- ('48'H)
r3PDSPDaoxxn(73), -- ('49'H)
r3DPSDoax(74), -- ('4A'H)
r3PDSnox(75), -- ('4B'H)
r3SDPana(76), -- ('4C'H)
r3SSPxDSxoxn(77), -- ('4D'H)
r3PDSPxox(78), -- ('4E'H)
r3PDSnoan(79), -- ('4F'H)
r3PDna(80), -- ('50'H)
r3DSPnaon(81), -- ('51'H)
r3DPSDaox(82), -- ('52'H)
r3SPDSxaxn(83), -- ('53'H)
r3DPSonon(84), -- ('54'H)
r3Dn(85), -- ('55'H)
r3DPSox(86), -- ('56'H)
r3DPSoan(87), -- ('57'H)
r3PDSPoax(88), -- ('58'H)
r3DPSnox(89), -- ('59'H)
r3DPx(90), -- ('5A'H)
r3DPSDonox(91), -- ('5B'H)
r3DPSDxox(92), -- ('5C'H)
r3DPSnoan(93), -- ('5D'H)
r3DPSDnaox(94), -- ('5E'H)
r3DPan(95), -- ('5F'H)
r3PDSxa(96), -- ('60'H)
r3DSPDSaoxxn(97), -- ('61'H)
r3DSPDoax(98), -- ('62'H)
r3SDPnox(99), -- ('63'H)
r3SDPSoax(100), -- ('64'H)
r3DSPnox(101), -- ('65'H)
r3DSx(102), -- ('66'H)
r3SDPSonox(103), -- ('67'H)
r3DSPDSonoxxn(104), -- ('68'H)
r3PDSxxn(105), -- ('69'H)
r3DPSax(106), -- ('6A'H)
r3PSDPSoaxxn(107), -- ('6B'H)
r3SDPax(108), -- ('6C'H)
r3PDSPDoaxxn(109), -- ('6D'H)
r3SDPSnoax(110), -- ('6E'H)
```

```
r3PDSxnan(111), -- ('6F'H)
r3PDSana(112), -- ('70'H)
r3SSDxPDxaxn(113), -- ('71'H)
r3SDPSxox(114), -- ('72'H)
r3SDPnoan(115), -- ('73'H)
r3DSPDxox(116), -- ('74'H)
r3DSPnoan(117), -- ('75'H)
r3SDPSnaox(118), -- ('76'H)
r3DSan(119), -- ('77'H)
r3PDSax(120), -- ('78'H)
r3DSPDSoaxxn(121), -- ('79'H)
r3DPSDnoax(122), -- ('7A'H)
r3SDPxnan(123), -- ('7B'H)
r3SPDSnoax(124), -- ('7C'H)
r3DPSxnan(125), -- ('7D'H)
r3SPxDSxo(126), -- ('7E'H)
r3DPSaan(127), -- ('7F'H)
r3DPSaa(128), -- ('80'H)
r3SPxDSxon(129), -- ('81'H)
r3DPSxna(130), -- ('82'H)
r3SPDSnoaxn(131), -- ('83'H)
r3SDPxna(132), -- ('84'H)
r3PDSPnoaxn(133), -- ('85'H)
r3DSPDSoaxx(134), -- ('86'H)
r3PDSaxn(135), -- ('87'H)
r3DSa(136), -- ('88'H)
r3SDPSnaoxn(137), -- ('89'H)
r3DSPnoa(138), -- ('8A'H)
r3DSPDxoxn(139), -- ('8B'H)
r3SDPnoa(140), -- ('8C'H)
r3SDPSxoxn(141), -- ('8D'H)
r3SSDxPDxax(142), -- ('8E'H)
r3PDSanan(143), -- ('8F'H)
r3PDSxna(144), -- ('90'H)
r3SDPSnoaxn(145), -- ('91'H)
r3DPSDPoaxx(146), -- ('92'H)
r3SPDaxn(147), -- ('93'H)
r3PSDPSoaxx(148), -- ('94'H)
r3DPSaxn(149), -- ('95'H)
r3DPSxx(150), -- ('96'H)
r3PSDPSonoxx(151), -- ('97'H)
r3SDPSonoxn(152), -- ('98'H)
r3DSxn(153), -- ('99'H)
r3DPSnax(154), -- ('9A'H)
r3SDPSoaxn(155), -- ('9B'H)
r3SPDnax(156), -- ('9C'H)
r3DSPDoaxn(157), -- ('9D'H)
r3DSPDSaoxx(158), -- ('9E'H)
r3PDSxan(159), -- ('9F'H)
r3DPa(160), -- ('A0'H)
r3PDSPnaoxn(161), -- ('A1'H)
r3DPSnoa(162), -- ('A2'H)
r3DPSDxoxn(163), -- ('A3'H)
r3PDSPonoxn(164), -- ('A4'H)
r3PDxn(165), -- ('A5'H)
r3DSPnax(166), -- ('A6'H)
r3PDSPoaxn(167), -- ('A7'H)
r3DPSoa(168), -- ('A8'H)
r3DPSoxn(169), -- ('A9'H)
r3D(170), -- ('AA'H)
r3DPSono(171), -- ('AB'H)
r3SPDSxax(172), -- ('AC'H)
r3DPSDaoxn(173), -- ('AD'H)
r3DSPnao(174), -- ('AE'H)
```

```
r3DPno(175), -- ('AF'H)
r3PDSnoa(176), -- ('B0'H)
r3PDSPxoxn(177), -- ('B1'H)
r3SSPxDSxox(178), -- ('B2'H)
r3SDPanan(179), -- ('B3'H)
r3PSDnax(180), -- ('B4'H)
r3DPSDoaxn(181), -- ('B5'H)
r3DPSDPaoxx(182), -- ('B6'H)
r3SDPxan(183), -- ('B7'H)
r3PSDPxax(184), -- ('B8'H)
r3DSPDaoxn(185), -- ('B9'H)
r3DPSnao(186), -- ('BA'H)
r3DSno(187), -- ('BB'H)
r3SPDSanax(188), -- ('BC'H)
r3SDxPDxan(189), -- ('BD'H)
r3DPSxo(190), -- ('BE'H)
r3DPSano(191), -- ('BF'H)
r3PSa(192), -- ('C0'H)
r3SPDSnaoxn(193), -- ('C1'H)
r3SPDSonoxn(194), -- ('C2'H)
r3PSxn(195), -- ('C3'H)
r3SPDnoa(196), -- ('C4'H)
r3SPDSxoxn(197), -- ('C5'H)
r3SDPnax(198), -- ('C6'H)
r3PSDPoaxn(199), -- ('C7'H)
r3SDPoa(200), -- ('C8'H)
r3SPDoxn(201), -- ('C9'H)
r3DPSDxax(202), -- ('CA'H)
r3SPDSaoxn(203), -- ('CB'H)
r3S(204), -- ('CC'H)
r3SDPono(205), -- ('CD'H)
r3SDPnao(206), -- ('CE'H)
r3SPno(207), -- ('CF'H)
r3PSDnoa(208), -- ('D0'H)
r3PSDPxoxn(209), -- ('D1'H)
r3PDSnax(210), -- ('D2'H)
r3SPDSoaxn(211), -- ('D3'H)
r3SSPxPDxax(212), -- ('D4'H)
r3DPSanan(213), -- ('D5'H)
r3PSDPSaoxx(214), -- ('D6'H)
r3DPSxan(215), -- ('D7'H)
r3PDSPxax(216), -- ('D8'H)
r3SDPSaoxn(217), -- ('D9'H)
r3DPSDanax(218), -- ('DA'H)
r3SPxDSxan(219), -- ('DB'H)
r3SPDnao(220), -- ('DC'H)
r3SDno(221), -- ('DD'H)
r3SDPxo(222), -- ('DE'H)
r3SDPano(223), -- ('DF'H)
r3PDSoa(224), -- ('E0'H)
r3PDSoxn(225), -- ('E1'H)
r3DSPDxax(226), -- ('E2'H)
r3PSDPaoxn(227), -- ('E3'H)
r3SDPSxax(228), -- ('E4'H)
r3PDSPaoxn(229), -- ('E5'H)
r3SDPSanax(230), -- ('E6'H)
r3SPxPDxan(231), -- ('E7'H)
r3SSPxDSxax(232), -- ('E8'H)
r3DSPDSanaxxn(233), -- ('E9'H)
r3DPSao(234), -- ('EA'H)
r3DPSxno(235), -- ('EB'H)
r3SDPao(236), -- ('EC'H)
r3SDPxno(237), -- ('ED'H)
r3DSo(238), -- ('EE'H)
```

```
  r3SDPnoo(239), -- ('EF'H)
  r3P(240), -- ('F0'H)
  r3PDSono(241), -- ('F1'H)
  r3PDSnao(242), -- ('F2'H)
  r3PSno(243), -- ('F3'H)
  r3PSDnao(244), -- ('F4'H)
  r3PDno(245), -- ('F5'H)
  r3PDSxo(246), -- ('F6'H)
  r3PDSano(247), -- ('F7'H)
  r3PDSao(248), -- ('F8'H)
  r3PDSxno(249), -- ('F9'H)
  r3DPo(250), -- ('FA'H)
  r3DPSnoo(251), -- ('FB'H)
  r3PSo(252), -- ('FC'H)
  r3PSDnoo(253), -- ('FD'H)
  r3DPSoo(254), -- ('FE'H)
  r3WHITE(255) -- ('FF'H)--}(0..255)

SecondaryOrderType ::= INTEGER {
  cacheBitmapUncompressed(0), cacheColorTable(1), cacheBitmapCompressed(2)
}(0..255)

StreamID ::= INTEGER {
  streamLowPriority(1), streamMediumPriority(2), streamHighPriority(4)
}(0..255)

SynchronizeMessageType ::= INTEGER {synchronize(1)}(0..65535)

SystemPointerType ::= INTEGER {nullPointer(0), defaultPointer(32512)
 -- ('00007F00'H)--}(0..4294967295)

UpdateType ::= INTEGER {orders(0), bitmap(1), palette(2), synchronize(3)
}(0..65535)

WindowActivationAction ::= INTEGER {
  localWindowActive(1), hostedWindowActive(2), hostedWindowInvisible(3),
  pointerDeviceCapture(4),
  activateWindow(32769), -- ('8001'H)
  closeWindow(32770), -- ('8002'H)
  restoreWindow(32771), -- ('8003'H)
  windowManagerMenu(32772), -- ('8004'H)
  activationHelpKey(32785), -- ('8011'H)
  activationHelpIndexKey(32786), -- ('8012'H)
  activationHelpExtendedKey(32787) -- ('8013'H)--}(0..65535)

WindowAttribute ::= SEQUENCE {
  windowID          WindowID,
  windowExtra       Integer32,
  windowOwner       WindowID,
  windowFlags       WindowAttributeFlags,
  windowRectangle   Rectangle16
}

WindowListMessageType ::= INTEGER {updateWindowList(1)}(0..65535)

WindowTitle ::= CHOICE {
  noTitle       Integer8(255),
  titleString   T50String(SIZE (1..maxTitleString))
}

-- Capability Types
CapabilitySetType ::= INTEGER {
  bitmapCacheCapabilitySet(4), bitmapCapabilitySet(2),
  colorCacheCapabilitySet(10), controlCapabilitySet(5),
```

```
  generalCapabilitySet(1), orderCapabilitySet(3), pointerCapabilitySet(8),
  activationCapabilitySet(7), shareCapabilitySet(9)}(0..65535)

GeneralCapabilitySet ::= SEQUENCE {
  capabilitySetType        CapabilitySetType(generalCapabilitySet),
  lengthCapability         Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  osMajorType              OSMajorType,
  osMinorType              OSMinorType,
  protocolVersion          Integer16(512), -- ('0200'H)
  pad2octetsA              Integer16,
  generalCompressionTypes  Integer16,
  pad2octetsB              Integer16,
  updatecapabilityFlag     Boolean16,
  remoteUnshareFlag        Boolean16,
  generalCompressionLevel  Integer16,
  pad2octetsC              Integer16
}

BitmapCapabilitySet ::= SEQUENCE {
  capabilitySetType         CapabilitySetType(bitmapCapabilitySet),
  lengthCapability          Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  preferredBitsPerPixel     Integer16(1..8),
  receive1BitPerPixelFlag   Boolean16,
  receive4BitsPerPixelFlag  Boolean16,
  receive8BitsPerPixelFlag  Boolean16,
  desktopWidth              Integer16,
  desktopHeight             Integer16,
  pad2octetsA               Integer16,
  desktopResizeFlag         Boolean16,
  bitmapCompressionType     BitmapCompressionCapabilityFlags,
  pad2octetsC               Integer16
}

OrderCapabilitySet ::= SEQUENCE {
  capabilitySetType    CapabilitySetType(orderCapabilitySet),
  lengthCapability     Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  terminalDescriptor   T50String(SIZE (1..maxTerminalDescriptor)),
  pad4octetsA          Integer32(0),
  desktopXGranularity  Integer16,
  desktopYGranularity  Integer16,
  pad2octetsA          Integer16(0),
  maximumOrderLevel    Integer16,
  numberFonts          Integer16(1..maxFonts),
  orderFlags           OrderCapabilityFlags,
  orderSupport
    SEQUENCE {destinationBltSupport     Integer8,
              patternBltSupport         Integer8,
              screenBltSupport          Integer8,
              memoryBltSupport          Integer8,
              memoryThreeWayBltSupport  Integer8,
              textSupport               Integer8,
              extendedTextSupport       Integer8,
              rectangleSupport          Integer8,
              lineSupport               Integer8,
              frameSupport              Integer8,
              opaqueRectangleSupport    Integer8,
              desktopSaveSupport        Integer8,
              undefinedOrder12          Integer8(0),
```

```
                undefinedOrder13          Integer8(0),
                undefinedOrder14          Integer8(0),
                undefinedOrder15          Integer8(0),
                undefinedOrder16          Integer8(0),
                undefinedOrder17          Integer8(0),
                undefinedOrder18          Integer8(0),
                undefinedOrder19          Integer8(0),
                undefinedOrder20          Integer8(0),
                undefinedOrder21          Integer8(0),
                undefinedOrder22          Integer8(0),
                undefinedOrder23          Integer8(0),
                undefinedOrder24          Integer8(0),
                undefinedOrder25          Integer8(0),
                undefinedOrder26          Integer8(0),
                undefinedOrder27          Integer8(0),
                undefinedOrder28          Integer8(0),
                undefinedOrder29          Integer8(0),
                undefinedOrder30          Integer8(0),
                undefinedOrder31          Integer8(0)},
  textFlags             TextCapabilityFlags,
  pad2octetsB           Integer16(0),
  pad4octetsB           Integer32(0),
  desktopSaveSize       Integer32,
  pad4octetsC           Integer32(0)
}

BitmapCacheCapabilitySet ::= SEQUENCE {
  capabilitySetType      CapabilitySetType(bitmapCacheCapabilitySet),
  lengthCapability       Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  pad4octetsA            Integer32(0),
  pad4octetsB            Integer32(0),
  pad4octetsC            Integer32(0),
  pad4octetsD            Integer32(0),
  pad4octetsE            Integer32(0),
  pad4octetsF            Integer32(0),
  cache1Entries          Integer16,
  cache1MaximumCellSize  Integer16(256..16384),
  cache2Entries          Integer16,
  cache2MaximumCellSize  Integer16(256..16384),
  cache3Entries          Integer16,
  cache3MaximumCellSize  Integer16(256..16384)
}

ColorCacheCapabilitySet ::= SEQUENCE {
  capabilitySetType    CapabilitySetType(colorCacheCapabilitySet),
  lengthCapability     Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  colorTablecacheSize  Integer16(1..255),
  pad2octetsA          Integer16
}

ActivationCapabilitySet ::= SEQUENCE {
  capabilitySetType    CapabilitySetType(activationCapabilitySet),
  lengthCapability     Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  helpKeyFlag          Boolean16,
  helpIndexKeyFlag     Boolean16,
  helpExtendedKeyFlag  Boolean16,
  windowActivateFlag   Boolean16
}
```

```
ControlCapabilitySet ::= SEQUENCE {
  capabilitySetType  CapabilitySetType(controlCapabilitySet),
  lengthCapability   Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  controlFlags       ControlCapabilityFlags,
  remoteDetachFlag   Boolean16,
  controlInterest    ControlPriority,
  detachInterest     ControlPriority
}

PointerCapabilitySet ::= SEQUENCE {
  capabilitySetType  CapabilitySetType(pointerCapabilitySet),
  lengthCapability   Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  colorPointerFlag   Boolean16,
  pointerCacheSize   Integer16(1..500)
}

ShareCapabilitySet ::= SEQUENCE {
  capabilitySetType  CapabilitySetType(shareCapabilitySet),
  lengthCapability   Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  nodeID             Integer32
}

NonStandardCapabilitySet ::= SEQUENCE {
  capabilitySetType      Integer16,
  -- defined by ASCE
  lengthCapability       Integer16,
  -- length of capability set in octets
  -- (including type and length parameters)
  nonStandardParameters  OCTET STRING
}

CombinedCapabilities ::= SEQUENCE {
  numberCapabilities    Integer16,
  -- number of capabilities in combinedCapabilities set
  pad2octets            Integer16(0),
  combinedCapabilities
    SET {generalCapabilitySet      GeneralCapabilitySet,
         bitmapCapabilitySet       BitmapCapabilitySet,
         orderCapabilitySet        OrderCapabilitySet,
         bitmapCacheCapabilitySet  BitmapCacheCapabilitySet,
         colorCacheCapabilitySet   ColorCacheCapabilitySet,
         activationCapabilitySet   ActivationCapabilitySet,
         controlCapabilitySet      ControlCapabilitySet,
         pointerCapabilitySet      PointerCapabilitySet,
         shareCapabilitySet        ShareCapabilitySet,
         nonStandardCapabilitySet  NonStandardCapabilitySet OPTIONAL}
}
UpdateCapabilitySet ::= CHOICE {bitmapCapabilitySet  BitmapCapabilitySet
}

-- Input Types
InputEvent ::= CHOICE {
  pointingDeviceEvent  PointingDeviceEvent,
  keyboardEvent        KeyboardEvent,
  synchronizeEvent     SynchronizeEvent
}
```

```
KeyboardEvent ::= SEQUENCE {
  eventTime      Integer32,
  messageType    InputMessageType(inputCodePoint | inputVirtualKey),
  keyboardFlags  KeyboardFlags,
  keyCode        Integer16
  -- AS protocol code page codepoint or virtual keycode
}

PointingDeviceEvent ::= SEQUENCE {
  eventTime            Integer32,
  messageType          InputMessageType(inputPointingDevice),
  pointingDeviceFlags  PointingDeviceFlags,
  pointingDeviceX      Coordinate16,
  pointingDeviceY      Coordinate16
}

SynchronizeEvent ::= SEQUENCE {
  eventTime    Integer32,
  messageType  InputMessageType(inputSynchronize)
}

-- Common Header Types
PrimaryOrderHeader ::= SEQUENCE {
  controlFlags   ControlOrderFlags,
  orderType      PrimaryOrderType OPTIONAL,
  encodingFlags  SEQUENCE (SIZE (1..3)) OF BitString8,
  boundsFlags    BoundsOrderFlags OPTIONAL,
  boundsLeft     Coordinate OPTIONAL,
  boundsTop      Coordinate OPTIONAL,
  boundsRight    Coordinate OPTIONAL,
  boundsBottom   Coordinate OPTIONAL
}

SecondaryOrderHeader ::= SEQUENCE {
  controlFlags  ControlOrderFlags,
  orderLength   Integer16,
  -- length in octets, from and including orderType, minus eight
  extraFlags    ExtraOrderFlags,
  orderType     SecondaryOrderType
}

ShareControlHeader ::= SEQUENCE {
  totalLength      Integer16(0..32767),
  protocolVersion  Integer4(1),
  pduType          PDUType,
  pad1octet        Integer8(0),
  pduSource        UserID
}

ShareDataHeader ::= SEQUENCE {
  shareControlHeader       ShareControlHeader, -- PDUType = dataPDU
  shareID                  ShareID,
  pad1octet                Integer8(0),
  streamID                 StreamID,
  uncompressedLength       Integer16,
  pduType2                 PDUType2,
  generalCompressedType    Integer8,
  generalCompressedLength  Integer16
}

-- Order Types

DestinationBltOrder ::= SEQUENCE {
  header      PrimaryOrderHeader, -- PrimaryOrderType = destinationBlt
```

```
  destLeft     Coordinate OPTIONAL,
  destTop      Coordinate OPTIONAL,
  destWidth    Coordinate OPTIONAL,
  destHeight   Coordinate OPTIONAL,
  rop3         ROP3 OPTIONAL
}

PatternBltOrder ::= SEQUENCE {
  header           PrimaryOrderHeader, -- PrimaryOrderType = patternBlt
  destLeft         Coordinate OPTIONAL,
  destTop          Coordinate OPTIONAL,
  destWidth        Coordinate OPTIONAL,
  destHeight       Coordinate OPTIONAL,
  rop3             ROP3 OPTIONAL,
  backgroundColor  Color OPTIONAL,
  foregroundColor  Color OPTIONAL,
  brush            Brush OPTIONAL
}

ScreenBltOrder ::= SEQUENCE {
  header       PrimaryOrderHeader, -- PrimaryOrderType = screenBlt
  destLeft     Coordinate OPTIONAL,
  destTop      Coordinate OPTIONAL,
  destWidth    Coordinate OPTIONAL,
  destHeight   Coordinate OPTIONAL,
  rop3         ROP3 OPTIONAL,
  sourceX      Coordinate OPTIONAL,
  sourceY      Coordinate OPTIONAL
}

CacheBitmapOrder ::= SEQUENCE {
  header            SecondaryOrderHeader, -- SecondaryOrderType =
  -- cacheBitmapUncompressed | cacheBitmapCompressed
  cacheId           Integer8(0..2),
  pad1octet         Integer8(0),
  bitmapWidth       Integer8,
  bitmapHeight      Integer8,
  bitmapBitsPerPel  Integer8(1 | 4 | 8),
  bitmapLength      Integer16,
  -- length of bitmapData in octets (after any compression)
  cacheIndex        Integer16,
  bitmapData        BitmapData
}

CacheColorTableOrder ::= SEQUENCE {
  header        SecondaryOrderHeader, -- SecondaryOrderType = cacheColorTable
  cacheIndex    Integer8,
  numberColors  Integer16(16 | 256),
  colorTable    SEQUENCE (SIZE (16 | 256)) OF ColorQuad
}

MemoryBltOrder ::= SEQUENCE {
  header                PrimaryOrderHeader, -- PrimaryOrderType = memoryBlt
  colorTableCacheIndex  Integer8 OPTIONAL,
  bitmapCacheID         Integer8 OPTIONAL,
  destLeft              Coordinate OPTIONAL,
  destTop               Coordinate OPTIONAL,
  destWidth             Coordinate OPTIONAL,
  destHeight            Coordinate OPTIONAL,
  rop3                  ROP3 OPTIONAL,
  sourceX               Coordinate OPTIONAL,
  sourceY               Coordinate OPTIONAL,
  bitmapCacheIndex      Integer16 OPTIONAL
}
```

```
MemoryThreeWayBltOrder ::= SEQUENCE {
  header                PrimaryOrderHeader,
  -- PrimaryOrderType = memoryThreeWayBlt
  colorTableCacheIndex  Integer8 OPTIONAL,
  bitmapCacheID         Integer8 OPTIONAL,
  destLeft              Coordinate OPTIONAL,
  destTop               Coordinate OPTIONAL,
  destWidth             Coordinate OPTIONAL,
  destHeight            Coordinate OPTIONAL,
  rop3                  ROP3 OPTIONAL,
  sourceX               Coordinate OPTIONAL,
  sourceY               Coordinate OPTIONAL,
  backgroundColor       Color OPTIONAL,
  foregroundColor       Color OPTIONAL,
  brush                 Brush OPTIONAL,
  bitmapCacheIndex      Integer16 OPTIONAL
}

TextOrder ::= SEQUENCE {
  header              PrimaryOrderHeader, -- PrimaryOrderType = text
  backMixMode         BackgroundMixMode OPTIONAL,
  startX              Coordinate OPTIONAL,
  startY              Coordinate OPTIONAL,
  backgroundColor     Color OPTIONAL,
  foregroundColor     Color OPTIONAL,
  extraSpacing        Integer16 OPTIONAL,
  totalBreakSpacing   Integer16 OPTIONAL,
  breakCount          Integer16 OPTIONAL,
  fontHeight          Integer16 OPTIONAL,
  fontWidth           Integer16 OPTIONAL,
  fontWeight          Integer16 OPTIONAL,
  textFlags           TextAttributeFlags OPTIONAL,
  fontID              Integer16 OPTIONAL,
  numberCodePoints    Integer8(1..255) OPTIONAL,
  -- number of codepoints in codePointList
  codePointList       ASString(SIZE (1..255)) OPTIONAL
}

ExtendedTextOrder ::= SEQUENCE {
  header              PrimaryOrderHeader, -- PrimaryOrderType = extendedText
  backMixMode         BackgroundMixMode OPTIONAL,
  startX              Coordinate OPTIONAL,
  startY              Coordinate OPTIONAL,
  backgroundColor     Color OPTIONAL,
  foregroundColor     Color OPTIONAL,
  extraSpacing        Integer16 OPTIONAL,
  totalBreakSpacing   Integer16 OPTIONAL,
  breakCount          Integer16 OPTIONAL,
  fontHeight          Integer16 OPTIONAL,
  fontWidth           Integer16 OPTIONAL,
  fontWeight          Integer16 OPTIONAL,
  textFlags1          TextAttributeFlags OPTIONAL,
  fontID              Integer16 OPTIONAL,
  textFlags2          ExtraTextFlags OPTIONAL,
  clipLeft            Coordinate OPTIONAL,
  clipTop             Coordinate OPTIONAL,
  clipRight           Coordinate OPTIONAL,
  clipBottom          Coordinate OPTIONAL,
  numberCodePoints    Integer8(1..255) OPTIONAL,
  -- number of codepoints in codePointList; where deltaX values
  -- are present maximum number of codepoints is 127
  codePointList       ASString(SIZE (1..255)) OPTIONAL,
  numberDeltaX        Integer8(1..127) OPTIONAL,
```

```
  -- number of deltaX values in deltaXList
  deltaXList          SEQUENCE (SIZE (1..127)) OF Coordinate OPTIONAL
}

FrameOrder ::= SEQUENCE {
  header              PrimaryOrderHeader, -- PrimaryOrderType = frame
  destLeft            Coordinate OPTIONAL,
  destTop             Coordinate OPTIONAL,
  destWidth           Coordinate OPTIONAL,
  destHeight          Coordinate OPTIONAL,
  rop3                ROP3 OPTIONAL,
  backgroundColor     Color OPTIONAL,
  foregroundColor     Color OPTIONAL,
  brush               Brush OPTIONAL
}

RectangleOrder ::= SEQUENCE {
  header              PrimaryOrderHeader, -- PrimaryOrderType = rectangle
  backMixMode         BackgroundMixMode OPTIONAL,
  destLeft            Coordinate OPTIONAL,
  destTop             Coordinate OPTIONAL,
  destRight           Coordinate OPTIONAL,
  destBottom          Coordinate OPTIONAL,
  backgroundColor     Color OPTIONAL,
  foregroundColor     Color OPTIONAL,
  brush               Brush OPTIONAL,
  rop2                ROP2 OPTIONAL,
  pen                 Pen OPTIONAL
}

OpaqueRectangleOrder ::= SEQUENCE {
  header       PrimaryOrderHeader, -- PrimaryOrderType = opaqueRectangle
  destLeft     Coordinate OPTIONAL,
  destTop      Coordinate OPTIONAL,
  destWidth    Coordinate OPTIONAL,
  destHeight   Coordinate OPTIONAL,
  color        Color OPTIONAL
}

LineOrder ::= SEQUENCE {
  header              PrimaryOrderHeader, -- PrimaryOrderType = line
  backMixMode         BackgroundMixMode OPTIONAL,
  startX              Coordinate OPTIONAL,
  startY              Coordinate OPTIONAL,
  endX                Coordinate OPTIONAL,
  endY                Coordinate OPTIONAL,
  backgroundColor     Color OPTIONAL,
  rop2                ROP2 OPTIONAL,
  pen                 Pen OPTIONAL
}

DesktopSaveOrder ::= SEQUENCE {
  header       PrimaryOrderHeader, -- PrimaryOrderType = desktopSave
  saveOffset   Integer32 OPTIONAL,
  destLeft     Coordinate OPTIONAL,
  destTop      Coordinate OPTIONAL,
  destWidth    Coordinate OPTIONAL,
  destHeight   Coordinate OPTIONAL,
  action       DesktopSaveAction OPTIONAL
}

DesktopOriginOrder ::= SEQUENCE {
  header        PrimaryOrderHeader, -- PrimaryOrderType = desktopOrigin
  desktopLeft   Coordinate OPTIONAL,
```

```
  desktopTop   Coordinate OPTIONAL
}

PrimaryOrder ::= CHOICE {
  destinationBlt     DestinationBltOrder,
  patternBlt         PatternBltOrder,
  screenBlt          ScreenBltOrder,
  memoryBlt          MemoryBltOrder,
  memoryThreeWayBlt  MemoryThreeWayBltOrder,
  text               TextOrder,
  extendedText       ExtendedTextOrder,
  frame              FrameOrder,
  rectangle          RectangleOrder,
  line               LineOrder,
  opaqueRectangle    OpaqueRectangleOrder,
  desktopSave        DesktopSaveOrder,
  desktopOrigin      DesktopOriginOrder
}

SecondaryOrder ::= CHOICE {
  cacheBitmap      CacheBitmapOrder,
  cacheColorTable  CacheColorTableOrder
}

UpdateOrder ::= CHOICE {
  primaryOrder    PrimaryOrder,
  secondaryOrder  SecondaryOrder
}

--/////////////////////////////////////////////////////////
--
--              Begin AS PDU Definitions
--
--/////////////////////////////////////////////////////////

ApplicationPDU ::= SEQUENCE {
  shareDataHeader     ShareDataHeader, -- PDUType2 = application
  action              ApplicationAction,
  numberApplications  Integer16,
  windowID            WindowID
}

ConfirmActivePDU ::= SEQUENCE {
  shareControlHeader          ShareControlHeader, -- PDUType = confirmActivePDU
  shareID                     ShareID,
  originatorID                UserID,
  lengthSourceDescriptor      Integer16(1..maxSourceDescriptor),
  -- length of sourceDescriptor in octets
  -- (including null terminator)
  lengthCombinedCapabilities  Integer16,
  -- length of combinedCapabilities in octets
  sourceDescriptor            T50String(SIZE (1..maxSourceDescriptor)),
  combinedCapabilities        CombinedCapabilities
}
ControlPDU ::= SEQUENCE {
  shareDataHeader  ShareDataHeader, -- PDUType2 = control
  action           ControlAction,
  grantID          UserID,
  controlID        Integer32(0..2147483647)
}

DeactivateAllPDU ::= SEQUENCE {
  shareControlHeader      ShareControlHeader, -- PDUType = deactivateAllPDU
  shareID                 ShareID,
```

```
  lengthSourceDescriptor  Integer16(1..maxSourceDescriptor),
  -- length of sourceDescriptor in octets
  -- (including null terminator)
  sourceDescriptor        T50String(SIZE (1..maxSourceDescriptor))
}

DeactivateOtherPDU ::= SEQUENCE {
  shareControlHeader      ShareControlHeader, -- PDUType = deactivateOtherPDU
  shareID                 ShareID,
  deactivateID            UserID,
  lengthSourceDescriptor  Integer16(1..maxSourceDescriptor),
  -- length of sourceDescriptor in octets
  -- (including null terminator)
  sourceDescriptor        T50String(SIZE (1..maxSourceDescriptor))
}

DeactivateSelfPDU ::= SEQUENCE {
  shareControlHeader  ShareControlHeader, -- PDUType = deactivateSelfPDU
  shareID             ShareID
}

DemandActivePDU ::= SEQUENCE {
  shareControlHeader          ShareControlHeader, -- PDUType = demandActivePDU
  shareID                     ShareID,
  lengthSourceDescriptor      Integer16(1..maxSourceDescriptor),
  -- length of sourceDescriptor in octets
  -- (including null terminator)
  lengthCombinedCapabilities  Integer16,
  -- length of combinedCapabilities in octets
  sourceDescriptor            T50String(SIZE (1..maxSourceDescriptor)),
  combinedCapabilities        CombinedCapabilities
}

FlowPDU ::= SEQUENCE {
  flowMarker      Integer16(32768), -- ('8000'H),
  -- distinguishes FlowPDUs from ASPDUs
  -- containing ShareControlHeaders
  pad8bits        Integer8(0),
  pduTypeFlow     PDUTypeFlow(flowResponsePDU | flowStopPDU | flowTestPDU),
  flowIdentifier  Integer8(0..127),
  flowNumber      Integer8,
  -- shall be zero for PDUType FlowStopPDU
  pduSource       UserID
  -- MCS User ID of sending ASCE
}

FontPDU ::= SEQUENCE {
  shareDataHeader  ShareDataHeader, -- PDUType2 = font
  numberFonts      Integer16(1..maxFonts),
  -- number of FontAttributes in fontList
  entrySize        Integer16,
  fontList         SEQUENCE (SIZE (1..maxFonts)) OF FontAttribute
}

InputPDU ::= SEQUENCE {
  shareDataHeader  ShareDataHeader, -- PDUType2 = input
  numberEvents     Integer16,
  -- number of InputEvents in eventList
  pad2octets       Integer16(0),
  eventList        SEQUENCE (SIZE (1..maxInputEvents)) OF InputEvent
}

MediatedControlPDU ::= SEQUENCE {
  shareDataHeader      ShareDataHeader, -- PDUType2 = mediatedControl
```

```
  action                MediatedControlAction,
  passControlFlag       Boolean16,
  sendingReference      Integer16,
  originatorReference   Integer16,
  originatorID          UserID
}

PointerPDU ::= SEQUENCE {
  shareDataHeader  ShareDataHeader, -- PDUType2 = pointer
  messageType      PointerMessageType,
  pad2octets       Integer16(0),
  pointerData
    CHOICE {systemPointerType      SystemPointerType,
            monoPointerAttribute   MonoPointerAttribute,
            colorPointerAttribute  ColorPointerAttribute,
            cachedPointerIndex     Integer16,
            pointerPosition        Point16}
}

RemoteSharePDU ::= SEQUENCE {
  shareDataHeader    ShareDataHeader, -- PDUType2 = remoteShare
  action             RemoteShareAction,
  additionalData
    CHOICE {requestingID  UserID,
            pad2octets    Integer16(0),
            denialCode    RemoteShareDenial},
  encryptedPassword  OCTET STRING(SIZE (1..maxPassword))
}

RequestActivePDU ::= SEQUENCE {
  shareControlHeader          ShareControlHeader, -- PDUType = requestActivePDU
  lengthSourceDescriptor      Integer16(1..maxSourceDescriptor),
  -- length of sourceDescriptor in octets
  -- (including null terminator)
  lengthCombinedCapabilities  Integer16,
  -- length of combinedCapabilities in octets
  sourceDescriptor            T50String(SIZE (1..maxSourceDescriptor)),
  combinedCapabilities        CombinedCapabilities
}

SynchronizePDU ::= SEQUENCE {
  shareDataHeader  ShareDataHeader, -- PDUType2 = synchronize
  messageType      SynchronizeMessageType,
  targetUser       UserID
}

UpdateBitmapPDU ::= SEQUENCE {
  shareDataHeader  ShareDataHeader, -- PDUType2=update
  updateType       UpdateType(bitmap),
  pad2octets       Integer16(0),
  destLeft         Coordinate16,
  destTop          Coordinate16,
  destRight        Coordinate16,
  destBottom       Coordinate16,
  width            Integer16,
  height           Integer16,
  bitsPerPixel     Integer16(1 | 4 | 8),
  compressedFlag   Boolean16,
  bitmapLength     Integer16,
  -- length in octets of bitmapData (after any compression)
  bitmapData       BitmapData
}

UpdateCapabilityPDU ::= SEQUENCE {
```

```
  shareDataHeader       ShareDataHeader, -- PDUType2 = updateCapability
  updateCapabilitySet   UpdateCapabilitySet
}

UpdateOrdersPDU ::= SEQUENCE {
  shareDataHeader   ShareDataHeader, -- PDUType2 = update
  updateType        UpdateType(orders),
  pad2octetsA       Integer16(0),
  numberOrders      Integer16,
  -- number of UpdateOrders in orderList
  pad2octetsB       Integer16(0),
  orderList         SEQUENCE OF UpdateOrder
}

UpdatePalettePDU ::= SEQUENCE {
  shareDataHeader   ShareDataHeader, -- PDUType2 = update
  updateType        UpdateType(palette),
  pad2octets        Integer16(0),
  numberColors      Integer32(16 | 256),
  palette           SEQUENCE (SIZE (16 | 256)) OF Color
}

UpdateSynchronizePDU ::= SEQUENCE {
  shareDataHeader   ShareDataHeader, -- PDUType2 = update
  updateType        UpdateType(synchronize),
  pad2octets        Integer16(0)
}

WindowActivationPDU ::= SEQUENCE {
  shareDataHeader    ShareDataHeader, -- PDUType2 = windowActivation
  action             WindowActivationAction,
  activationID       Integer16,
  activationWindow   WindowID,
  activationPoint    Point16
}

WindowListPDU ::= SEQUENCE {
  shareDataHeader       ShareDataHeader, -- PDUType2 = windowList
  messageType           WindowListMessageType,
  pad2octetsA           Integer16,
  numberWindows         Integer16,
  -- number of WindowAttributes/Titles in lists
  listTime              Integer16,
  listID                Integer16,
  pad2octetsB           Integer16,
  windowAttributeList   SEQUENCE OF WindowAttribute,
  windowTitleList       SEQUENCE OF WindowTitle
}

SharePDU ::= CHOICE {
  applicationPDU         ApplicationPDU,
  confirmActivePDU       ConfirmActivePDU,
  controlPDU             ControlPDU,
  deactivateAllPDU       DeactivateAllPDU,
  deactivateOtherPDU     DeactivateOtherPDU,
  deactivateSelfPDU      DeactivateSelfPDU,
  demandActivePDU        DemandActivePDU,
  flowPDU                FlowPDU,
  fontPDU                FontPDU,
  inputPDU               InputPDU,
  mediatedControlPDU     MediatedControlPDU,
  pointerPDU             PointerPDU,
  remoteSharePDU         RemoteSharePDU,
  requestActivePDU       RequestActivePDU,
```

```
  synchronizePDU          SynchronizePDU,
  updateCapabilityPDU     UpdateCapabilityPDU,
  updateBitmapPDU         UpdateBitmapPDU,
  updateOrdersPDU         UpdateOrdersPDU,
  updateSynchronizePDU    UpdateSynchronizePDU,
  updatePalettePDU        UpdatePalettePDU,
  windowActivationPDU     WindowActivationPDU,
  windowListPDU           WindowListPDU
}

--/////////////////////////////////////////////////////////
--
--                  End AS Definitions
--
--/////////////////////////////////////////////////////////
END
```

9.2 Base mode ASN.1 definition

```
--/////////////////////////////////////////////////////////////////////////
--
--                     Begin AS Definitions
--
-- The following base mode ASN.1 definitions are encoded using the
-- BASIC ALIGNED variant of the Packed Encoding Rules of Recommendation
-- ITU-T X.691.
--
--/////////////////////////////////////////////////////////////////////////
AS-PROTOCOL-for-PER-encoding {itu-t(0) recommendation(0) t(20)
t128(128) version(0) 2 as-protocol-for-per-encoding(1)}
DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
IMPORTS
  H221NonStandardIdentifier, Key, NonStandardParameter, UserID
    FROM GCC-PROTOCOL {itu-t(0) recommendation(0) t(20) t124(124)
 version(0) 2 asn1Modules(2) gcc-protocol(1)};

-- NOTE: ===================================================
-- NOTE: All abstract types defined shall be exported
-- NOTE: ===================================================
-- Base Types
Coordinate8 ::= INTEGER(-128..127)
Coordinate16 ::= INTEGER(-32768..32767)
Integer8 ::= INTEGER(0..255)
Integer12 ::= INTEGER(0..4095)
Integer16 ::= INTEGER(0..65535)
Integer32 ::= INTEGER(0..4294967295)
Signed16 ::= INTEGER(-32768..32767)
ShareID ::= Integer32
WindowID ::= Integer32
T50String ::= OCTET STRING(SIZE (0..255)) -- T.50 String

ASString ::= OCTET STRING(SIZE (0..255)) -- AS Protocol CodePage String

-- Bit Flag Types
ExtraTextFlags ::= BIT STRING {
  opaqueRectangle(1), clipToRectangle(2), deltaXPresent(15)}

FontAttributeFlags ::= BIT STRING {fixedPitch(0), fixedSize(1)}

KeyboardFlags ::= BIT STRING {right(0), quiet(12), down(14), release(15)}

PointingDeviceFlags ::= BIT STRING {
  move(11), button1(12), button2(13), button3(14), down(15)}
```

```
TextAttributeFlags ::= BIT STRING {
  italic(2), underline(3), strikeout(4), baselineStart(8)}

WindowAttributeFlags ::= BIT STRING {
  minimized(0), taggable(1), hosted(2), shadow(3), local(4), topmost(5),
  windowManagerMinimized(16), windowManagerInvisible(17)}

-- General Types
ActivateWindowRequest ::= SEQUENCE {
  activationWindow       WindowID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ActivationHelpKeyRequest ::= SEQUENCE {
  activationWindow       WindowID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ActivationHelpIndexKeyRequest ::= SEQUENCE {
  activationWindow       WindowID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ActivationHelpExtendedKeyRequest ::= SEQUENCE {
  activationWindow       WindowID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

BackgroundMixMode ::= CHOICE {
  transparent                    [1]  NULL,
  opaque                         [2]  NULL,
  nonStandardBackgroundMixMode  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

BitmapData ::= CHOICE {
  uncompressedBitmapData  [0]  OCTET STRING,
  compressedBitmapData    [2]  CompressedBitmapData,
  nonStandardBitmapData   NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

Brush ::= SEQUENCE {
  originX                Integer8 OPTIONAL,
  originY                Integer8 OPTIONAL,
  style                  BrushStyle OPTIONAL,
  hatch                  BrushHatch OPTIONAL,
  pattern                OCTET STRING(SIZE (7)) OPTIONAL,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

BrushHatch ::= CHOICE {
```

```
  style                   HatchStyle,
  patternZero             Integer8,
  nonStandardBrushHatch   NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

BrushStyle ::= CHOICE {
  solid                   [0]  NULL,
  null                    [1]  NULL,
  hatched                 [2]  NULL,
  pattern                 [3]  NULL,
  nonStandardBrushStyle   NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

CloseWindowRequest ::= SEQUENCE {
  activationWindow        WindowID,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

Color ::= SEQUENCE {
  c1  Integer8,
  -- either R of RGB or subject to capability negotiation.
  c2  Integer8,
  -- either G of RGB or subject to capability negotiation.
  c3  Integer8
  -- either B of RGB or subject to capability negotiation
}

ColorAccuracyEnhancementRGB ::= CHOICE {
  predefinedRGBSpace    CHOICE {nonStandardRGBSpace  NonStandardParameter,
                                ...},
  generalRGBParameters
    SEQUENCE {gamma             REAL(0..MAX) OPTIONAL,
              -- Gamma value of the color space
              colorTemperature  INTEGER(0..MAX) OPTIONAL,
              -- Color temperature of the white point assumed by
              -- the color space (in degrees Kelvin)
              primaries
                SEQUENCE {red    ColorCIExyChromaticity,
                          -- CIE xy chromaticity coordinate of the red primary
                          green  ColorCIExyChromaticity,
                          -- CIE xy chromaticity coordinate of the green primary
                          blue   ColorCIExyChromaticity
                          -- CIE xy chromaticity coordinate of the blue primary
              } OPTIONAL,
              ...},
  ...
}

ColorCIExyChromaticity ::= SEQUENCE {
  x  REAL(0..1), -- CIE normalized x component
  y  REAL(0..1) -- CIE normalized y component
}

ColorPalette ::= CHOICE {
  paletteRGB
    SEQUENCE {palette      SEQUENCE (SIZE (16 | 256)) OF ColorRGB,
              enhancement  ColorAccuracyEnhancementRGB OPTIONAL,
              ...},
```

```
  nonStandardPalette  NonStandardParameter,
  ...
}

ColorPointerAttribute ::= SEQUENCE {
  cacheIndex             Integer16,
  hotSpot                Point16,
  width                  Integer16,
  height                 Integer16,
  colorPointerData       OCTET STRING,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ColorRGB ::= SEQUENCE {red    Integer8,
                       green  Integer8,
                       blue   Integer8
}

ColorSpaceSpecifier ::= CHOICE {
  colorSpaceDefault      NULL,
  -- Default color space is RGB without accuracy enhancement
  colorSpaceRGB          ColorAccuracyEnhancementRGB,
  nonStandardColorSpace  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

CompressedBitmapData ::= SEQUENCE {
  mainBodySize           Integer16,
  rowSize                Integer16,
  uncompressedSize       Integer16,
  compressedBitmap       OCTET STRING,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ConfirmDetachResponse ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}
ConfirmRemoteShare ::= SEQUENCE {
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ConfirmTakeResponse ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}
```

```
ControlPriority ::= CHOICE {
  always                      [1]  NULL,
  never                       [2]  NULL,
  confirm                     [3]  NULL,
  nonStandardControlPriority  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

Cooperate ::= SEQUENCE {
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

Coordinate ::= CHOICE {
  absolute               Coordinate16,
  delta                  Coordinate8,
  nonStandardCoordinate  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

DesktopSaveAction ::= CHOICE {
  desktopSave                   [0]  NULL,
  desktopRestore                [1]  NULL,
  nonStandardDesktopSaveAction  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

DenyDetachResponse ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DenyPassResponse ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DenyTakeResponse ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DenyRemoteDetachResponse ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
```

```
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DenyRemoteShare ::= CHOICE {
  remoteShareDenial  RemoteShareDenial,
  nonStandardDenial  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

Detach ::= SEQUENCE {
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DetachRequest ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FontAttribute ::= SEQUENCE {
  faceName               T50String,
  fontFlags              FontAttributeFlags,
  averageWidth           Integer16,
  height                 Integer16,
  aspectX                Integer16,
  aspectY                Integer16,
  signature1             Integer8,
  signature2             Integer8,
  signature3             Integer16,
  codePage               FontCodePage,
  ascent                 Integer16,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FontCodePage ::= CHOICE {
  allCodePoints            [0]  NULL,
  coreCodePoints           [255]  NULL,
  nonStandardFontCodePage  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

GeneralCompressionSpecifier ::= CHOICE {
  v42bisCompression       V42bisCompression,
  nonStandardCompression  NonStandardParameter,
  ...
}

GrantControl ::= SEQUENCE {
  grantID                UserID,
  controlID              INTEGER(0..2147483647),
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
```

```
  -- Subject to capability negotiation.
  ...
}

HatchStyle ::= CHOICE {
  horizontal              [0]  NULL,
  vertical                [1]  NULL,
  forward                 [2]  NULL,
  backward                [3]  NULL,
  cross                   [4]  NULL,
  diagonal                [5]  NULL,
  nonStandardHatchStyle   NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

HostedWindowActiveIndication ::= SEQUENCE {
  activationID            Integer16,
  activationWindow        WindowID,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

HostedWindowInvisibleIndication ::= SEQUENCE {
  activationID            Integer16,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

LocalWindowActiveIndication ::= SEQUENCE {
  activationID            Integer16,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

MonoPointerAttribute ::= SEQUENCE {
  hotSpot                 Point16,
  width                   Integer16,
  height                  Integer16,
  monoPointerData         OCTET STRING,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

NotifyHostedApplications ::= SEQUENCE {
  numberApplications      Integer16,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

PassControlRequest ::= SEQUENCE {
  passControlFlag         BOOLEAN,
  sendingReference        Integer16,
  originatorID            UserID,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}
```

```
Pen ::= SEQUENCE {
  style                  PenStyle OPTIONAL,
  width                  Integer8(1) OPTIONAL,
  color                  Color OPTIONAL,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

PenStyle ::= CHOICE {
  solid                [0]  NULL,
  dashed               [1]  NULL,
  dotted               [2]  NULL,
  dash-dot             [3]  NULL,
  dash-dot-dot         [4]  NULL,
  null                 [5]  NULL,
  nonStandardPenStyle  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

Point16 ::= SEQUENCE {x  Coordinate16,
                      y  Coordinate16
}

PointerDeviceCaptureIndication ::= SEQUENCE {
  activationID           Integer16,
  activationWindow       WindowID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

Rectangle16 ::= SEQUENCE {
  left    Coordinate16,
  top     Coordinate16,
  right   Coordinate16,
  bottom  Coordinate16
}

RemoteDetachRequest ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

RemoteShareDenial ::= CHOICE {
  incorrectPassword                [1]  NULL,
  remoteShareNotEnabled            [2]  NULL,
  remoteShareInOperationIncoming   [3]  NULL,
  remoteShareInOperationOutgoing   [4]  NULL,
  nonStandardRemoteShareDenial     NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

RequestControl ::= SEQUENCE {
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}
```

```
RequestRemoteShare ::= SEQUENCE {
  requestingID            UserID,
  encryptedPassword       OCTET STRING,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

RestoreWindowRequest ::= SEQUENCE {
  activationWindow        WindowID,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ROP2 ::= INTEGER {
  r2BLACK(1), r2DPon(2), r2DPna(3), r2Pn(4), r2PDna(5), r2Dn(6), r2DPx(7),
  r2DPan(8), r2DPa(9), r2DPxn(10), r2D(11), r2DPno(12), r2P(13), r2PDno(14),
  r2DPo(15), r2WHITE(16)}(0..255)
ROP3 ::= INTEGER {
  r3BLACK(0), -- ('00'H)
  r3DPSoon(1), -- ('01'H)
  r3DPSona(2), -- ('02'H)
  r3PSon(3), -- ('03'H)
  r3SDPona(4), -- ('04'H)
  r3DPon(5), -- ('05'H)
  r3PDSxnon(6), -- ('06'H)
  r3PDSaon(7), -- ('07'H)
  r3SDPnaa(8), -- ('08'H)
  r3PDSxon(9), -- ('09'H)
  r3DPna(10), -- ('0A'H)
  r3PSDnaon(11), -- ('0B'H)
  r3SPna(12), -- ('0C'H)
  r3PDSnaon(13), -- ('0D'H)
  r3PDSonon(14), -- ('0E'H)
  r3Pn(15), -- ('0F'H)
  r3PDSona(16), -- ('10'H)
  r3DSon(17), -- ('11'H)
  r3SDPxnon(18), -- ('12'H)
  r3SDPaon(19), -- ('13'H)
  r3DPSxnon(20), -- ('14'H)
  r3DPSaon(21), -- ('15'H)
  r3PSDPSanaxx(22), -- ('16'H)
  r3SSPxDSxaxn(23), -- ('17'H)
  r3SPxPDxa(24), -- ('18'H)
  r3SDPSanaxn(25), -- ('19'H)
  r3PDSPaox(26), -- ('1A'H)
  r3SDPSxaxn(27), -- ('1B'H)
  r3PSDPaox(28), -- ('1C'H)
  r3DSPDxaxn(29), -- ('1D'H)
  r3PDSox(30), -- ('1E'H)
  r3PDSoan(31), -- ('1F'H)
  r3DPSnaa(32), -- ('20'H)
  r3SDPxon(33), -- ('21'H)
  r3DSna(34), -- ('22'H)
  r3SPDnaon(35), -- ('23'H)
  r3SPxDSxa(36), -- ('24'H)
  r3PDSPanaxn(37), -- ('25'H)
  r3SDPSaox(38), -- ('26'H)
  r3SDPSxnox(39), -- ('27'H)
  r3DPSxa(40), -- ('28'H)
  r3PSDPSaoxxn(41), -- ('29'H)
  r3DPSana(42), -- ('2A'H)
```

```
r3SSPxPDxaxn(43), -- ('2B'H)
r3SPDSoax(44), -- ('2C'H)
r3PSDnox(45), -- ('2D'H)
r3PSDPxox(46), -- ('2E'H)
r3PSDnoan(47), -- ('2F'H)
r3PSna(48), -- ('30'H)
r3SDPnaon(49), -- ('31'H)
r3SDPSoox(50), -- ('32'H)
r3Sn(51), -- ('33'H)
r3SPDSaox(52), -- ('34'H)
r3SPDSxnox(53), -- ('35'H)
r3SDPox(54), -- ('36'H)
r3SDPoan(55), -- ('37'H)
r3PSDPoax(56), -- ('38'H)
r3SPDnox(57), -- ('39'H)
r3SPDSxox(58), -- ('3A'H)
r3SPDnoan(59), -- ('3B'H)
r3PSx(60), -- ('3C'H)
r3SPDSonox(61), -- ('3D'H)
r3SPDSnaox(62), -- ('3E'H)
r3PSan(63), -- ('3F'H)
r3PSDnaa(64), -- ('40'H)
r3DPSxon(65), -- ('41'H)
r3SDxPDxa(66), -- ('42'H)
r3SPDSanaxn(67), -- ('43'H)
r3SDna(68), -- ('44'H)
r3DPSnaon(69), -- ('45'H)
r3DSPDaox(70), -- ('46'H)
r3PSDPxaxn(71), -- ('47'H)
r3SDPxa(72), -- ('48'H)
r3PDSPDaoxxn(73), -- ('49'H)
r3DPSDoax(74), -- ('4A'H)
r3PDSnox(75), -- ('4B'H)
r3SDPana(76), -- ('4C'H)
r3SSPxDSxoxn(77), -- ('4D'H)
r3PDSPxox(78), -- ('4E'H)
r3PDSnoan(79), -- ('4F'H)
r3PDna(80), -- ('50'H)
r3DSPnaon(81), -- ('51'H)
r3DPSDaox(82), -- ('52'H)
r3SPDSxaxn(83), -- ('53'H)
r3DPSonon(84), -- ('54'H)
r3Dn(85), -- ('55'H)
r3DPSox(86), -- ('56'H)
r3DPSoan(87), -- ('57'H)
r3PDSPoax(88), -- ('58'H)
r3DPSnox(89), -- ('59'H)
r3DPx(90), -- ('5A'H)
r3DPSDonox(91), -- ('5B'H)
r3DPSDxox(92), -- ('5C'H)
r3DPSnoan(93), -- ('5D'H)
r3DPSDnaox(94), -- ('5E'H)
r3DPan(95), -- ('5F'H)
r3PDSxa(96), -- ('60'H)
r3DSPDSaoxxn(97), -- ('61'H)
r3DSPDoax(98), -- ('62'H)
r3SDPnox(99), -- ('63'H)
r3SDPSoax(100), -- ('64'H)
r3DSPnox(101), -- ('65'H)
r3DSx(102), -- ('66'H)
r3SDPSonox(103), -- ('67'H)
r3DSPDSonoxxn(104), -- ('68'H)
r3PDSxxn(105), -- ('69'H)
r3DPSax(106), -- ('6A'H)
```

```
r3PSDPSoaxxn(107), -- ('6B'H)
r3SDPax(108), -- ('6C'H)
r3PDSPDoaxxn(109), -- ('6D'H)
r3SDPSnoax(110), -- ('6E'H)
r3PDSxnan(111), -- ('6F'H)
r3PDSana(112), -- ('70'H)
r3SSDxPDxaxn(113), -- ('71'H)
r3SDPSxox(114), -- ('72'H)
r3SDPnoan(115), -- ('73'H)
r3DSPDxox(116), -- ('74'H)
r3DSPnoan(117), -- ('75'H)
r3SDPSnaox(118), -- ('76'H)
r3DSan(119), -- ('77'H)
r3PDSax(120), -- ('78'H)
r3DSPDSoaxxn(121), -- ('79'H)
r3DPSDnoax(122), -- ('7A'H)
r3SDPxnan(123), -- ('7B'H)
r3SPDSnoax(124), -- ('7C'H)
r3DPSxnan(125), -- ('7D'H)
r3SPxDSxo(126), -- ('7E'H)
r3DPSaan(127), -- ('7F'H)
r3DPSaa(128), -- ('80'H)
r3SPxDSxon(129), -- ('81'H)
r3DPSxna(130), -- ('82'H)
r3SPDSnoaxn(131), -- ('83'H)
r3SDPxna(132), -- ('84'H)
r3PDSPnoaxn(133), -- ('85'H)
r3DSPDSoaxx(134), -- ('86'H)
r3PDSaxn(135), -- ('87'H)
r3DSa(136), -- ('88'H)
r3SDPSnaoxn(137), -- ('89'H)
r3DSPnoa(138), -- ('8A'H)
r3DSPDxoxn(139), -- ('8B'H)
r3SDPnoa(140), -- ('8C'H)
r3SDPSxoxn(141), -- ('8D'H)
r3SSDxPDxax(142), -- ('8E'H)
r3PDSanan(143), -- ('8F'H)
r3PDSxna(144), -- ('90'H)
r3SDPSnoaxn(145), -- ('91'H)
r3DPSDPoaxx(146), -- ('92'H)
r3SPDaxn(147), -- ('93'H)
r3PSDPSoaxx(148), -- ('94'H)
r3DPSaxn(149), -- ('95'H)
r3DPSxx(150), -- ('96'H)
r3PSDPSonoxx(151), -- ('97'H)
r3SDPSonoxn(152), -- ('98'H)
r3DSxn(153), -- ('99'H)
r3DPSnax(154), -- ('9A'H)
r3SDPSoaxn(155), -- ('9B'H)
r3SPDnax(156), -- ('9C'H)
r3DSPDoaxn(157), -- ('9D'H)
r3DSPDSaoxx(158), -- ('9E'H)
r3PDSxan(159), -- ('9F'H)
r3DPa(160), -- ('A0'H)
r3PDSPnaoxn(161), -- ('A1'H)
r3DPSnoa(162), -- ('A2'H)
r3DPSDxoxn(163), -- ('A3'H)
r3PDSPonoxn(164), -- ('A4'H)
r3PDxn(165), -- ('A5'H)
r3DSPnax(166), -- ('A6'H)
r3PDSPoaxn(167), -- ('A7'H)
r3DPSoa(168), -- ('A8'H)
r3DPSoxn(169), -- ('A9'H)
r3D(170), -- ('AA'H)
```

```
r3DPSono(171), -- ('AB'H)
r3SPDSxax(172), -- ('AC'H)
r3DPSDaoxn(173), -- ('AD'H)
r3DSPnao(174), -- ('AE'H)
r3DPno(175), -- ('AF'H)
r3PDSnoa(176), -- ('B0'H)
r3PDSPxoxn(177), -- ('B1'H)
r3SSPxDSxox(178), -- ('B2'H)
r3SDPanan(179), -- ('B3'H)
r3PSDnax(180), -- ('B4'H)
r3DPSDoaxn(181), -- ('B5'H)
r3DPSDPaoxx(182), -- ('B6'H)
r3SDPxan(183), -- ('B7'H)
r3PSDPxax(184), -- ('B8'H)
r3DSPDaoxn(185), -- ('B9'H)
r3DPSnao(186), -- ('BA'H)
r3DSno(187), -- ('BB'H)
r3SPDSanax(188), -- ('BC'H)
r3SDxPDxan(189), -- ('BD'H)
r3DPSxo(190), -- ('BE'H)
r3DPSano(191), -- ('BF'H)
r3PSa(192), -- ('C0'H)
r3SPDSnaoxn(193), -- ('C1'H)
r3SPDSonoxn(194), -- ('C2'H)
r3PSxn(195), -- ('C3'H)
r3SPDnoa(196), -- ('C4'H)
r3SPDSxoxn(197), -- ('C5'H)
r3SDPnax(198), -- ('C6'H)
r3PSDPoaxn(199), -- ('C7'H)
r3SDPoa(200), -- ('C8'H)
r3SPDoxn(201), -- ('C9'H)
r3DPSDxax(202), -- ('CA'H)
r3SPDSaoxn(203), -- ('CB'H)
r3S(204), -- ('CC'H)
r3SDPono(205), -- ('CD'H)
r3SDPnao(206), -- ('CE'H)
r3SPno(207), -- ('CF'H)
r3PSDnoa(208), -- ('D0'H)
r3PSDPxoxn(209), -- ('D1'H)
r3PDSnax(210), -- ('D2'H)
r3SPDSoaxn(211), -- ('D3'H)
r3SSPxPDxax(212), -- ('D4'H)
r3DPSanan(213), -- ('D5'H)
r3PSDPSaoxx(214), -- ('D6'H)
r3DPSxan(215), -- ('D7'H)
r3PDSPxax(216), -- ('D8'H)
r3SDPSaoxn(217), -- ('D9'H)
r3DPSDanax(218), -- ('DA'H)
r3SPxDSxan(219), -- ('DB'H)
r3SPDnao(220), -- ('DC'H)
r3SDno(221), -- ('DD'H)
r3SDPxo(222), -- ('DE'H)
r3SDPano(223), -- ('DF'H)
r3PDSoa(224), -- ('E0'H)
r3PDSoxn(225), -- ('E1'H)
r3DSPDxax(226), -- ('E2'H)
r3PSDPaoxn(227), -- ('E3'H)
r3SDPSxax(228), -- ('E4'H)
r3PDSPaoxn(229), -- ('E5'H)
r3SDPSanax(230), -- ('E6'H)
r3SPxPDxan(231), -- ('E7'H)
r3SSPxDSxax(232), -- ('E8'H)
r3DSPDSanaxxn(233), -- ('E9'H)
r3DPSao(234), -- ('EA'H)
```

```
  r3DPSxno(235), -- ('EB'H)
  r3SDPao(236), -- ('EC'H)
  r3SDPxno(237), -- ('ED'H)
  r3DSo(238), -- ('EE'H)
  r3SDPnoo(239), -- ('EF'H)
  r3P(240), -- ('F0'H)
  r3PDSono(241), -- ('F1'H)
  r3PDSnao(242), -- ('F2'H)
  r3PSno(243), -- ('F3'H)
  r3PSDnao(244), -- ('F4'H)
  r3PDno(245), -- ('F5'H)
  r3PDSxo(246), -- ('F6'H)
  r3PDSano(247), -- ('F7'H)
  r3PDSao(248), -- ('F8'H)
  r3PDSxno(249), -- ('F9'H)
  r3DPo(250), -- ('FA'H)
  r3DPSnoo(251), -- ('FB'H)
  r3PSo(252), -- ('FC'H)
  r3PSDnoo(253), -- ('FD'H)
  r3DPSoo(254), -- ('FE'H)
  r3WHITE(255) -- ('FF'H)--}(0..255)
SystemPointerType ::= CHOICE {
  null                              [0]  NULL,
  default                           [12512]  NULL,
  nonStandardSystemPointerValue  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

TakeControlRequest ::= SEQUENCE {
  passControlFlag        BOOLEAN,
  sendingReference       Integer16,
  originatorReference    Integer16,
  originatorID           UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

UnhostApplication ::= SEQUENCE {
  windowID               WindowID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

V42bisCompression ::= SEQUENCE {
  p1  INTEGER(512..65535) OPTIONAL,
  p2  INTEGER(6..250) OPTIONAL,
  ...
}

WindowAttribute ::= SEQUENCE {
  windowID               WindowID,
  windowExtra            Integer32,
  windowOwner            WindowID,
  windowFlags            WindowAttributeFlags,
  windowRectangle        Rectangle16,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

WindowManagerMenuRequest ::= SEQUENCE {
```

```
  activationWindow       WindowID,
  activationPoint        Point16,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

WindowTitle ::= CHOICE {
  noTitle                 Integer8(255),
  titleString             T50String,
  nonStandardWindowTitle  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

-- Input Types
InputEvent ::= CHOICE {
  pointingDeviceEvent    [12769]  PointingDeviceEvent,
  codePointEvent         [1]  CodePointEvent,
  virtualKeyEvent        [2]  VirtualKeyEvent,
  synchronizeEvent       [0]  SynchronizeEvent,
  nonStandardInputEvent  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

CodePointEvent ::= SEQUENCE {
  eventTime              Integer32,
  keyboardFlags          KeyboardFlags,
  codePoint              Integer16,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

VirtualKeyEvent ::= SEQUENCE {
  eventTime              Integer32,
  keyboardFlags          KeyboardFlags,
  virtualKey             Integer16,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

PointingDeviceEvent ::= SEQUENCE {
  eventTime              Integer32,
  pointingDeviceFlags    PointingDeviceFlags,
  pointingDeviceX        Coordinate16,
  pointingDeviceY        Coordinate16,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

SynchronizeEvent ::= SEQUENCE {
  eventTime              Integer32,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

-- Common Header Types
PrimaryOrderHeader ::= SEQUENCE {
  boundsLeft    Coordinate OPTIONAL,
```

```
  boundsTop      Coordinate OPTIONAL,
  boundsRight    Coordinate OPTIONAL,
  boundsBottom   Coordinate OPTIONAL,
  ...
}

ShareDataHeader ::= SEQUENCE {
  shareID                       ShareID,
  generalCompressionSpecifier   GeneralCompressionSpecifier OPTIONAL,
  ...
}

-- Order Types
DestinationBltOrder ::= SEQUENCE {
  header                 PrimaryOrderHeader,
  destLeft               Coordinate OPTIONAL,
  destTop                Coordinate OPTIONAL,
  destWidth              Coordinate OPTIONAL,
  destHeight             Coordinate OPTIONAL,
  rop3                   ROP3 OPTIONAL,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

PatternBltOrder ::= SEQUENCE {
  header                 PrimaryOrderHeader,
  destLeft               Coordinate OPTIONAL,
  destTop                Coordinate OPTIONAL,
  destWidth              Coordinate OPTIONAL,
  destHeight             Coordinate OPTIONAL,
  rop3                   ROP3 OPTIONAL,
  backgroundColor        Color OPTIONAL,
  foregroundColor        Color OPTIONAL,
  brush                  Brush OPTIONAL,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ScreenBltOrder ::= SEQUENCE {
  header                 PrimaryOrderHeader,
  destLeft               Coordinate OPTIONAL,
  destTop                Coordinate OPTIONAL,
  destWidth              Coordinate OPTIONAL,
  destHeight             Coordinate OPTIONAL,
  rop3                   ROP3 OPTIONAL,
  sourceX                Coordinate OPTIONAL,
  sourceY                Coordinate OPTIONAL,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

CacheBitmapOrder ::= SEQUENCE {
  cacheId                INTEGER(0..2),
  bitmapWidth            Integer8,
  bitmapHeight           Integer8,
  bitmapBitsPerPel       INTEGER(1 | 4 | 8),
  cacheIndex             Integer16,
  bitmapData             BitmapData,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
```

```
}

CacheColorTableOrder ::= SEQUENCE {
  cacheIndex              Integer8,
  colorTable              ColorPalette,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

MemoryBltOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  colorTableCacheIndex    Integer8 OPTIONAL,
  bitmapCacheID           Integer8 OPTIONAL,
  destLeft                Coordinate OPTIONAL,
  destTop                 Coordinate OPTIONAL,
  destWidth               Coordinate OPTIONAL,
  destHeight              Coordinate OPTIONAL,
  rop3                    ROP3 OPTIONAL,
  sourceX                 Coordinate OPTIONAL,
  sourceY                 Coordinate OPTIONAL,
  bitmapCacheIndex        Integer16 OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

MemoryThreeWayBltOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  colorTableCacheIndex    Integer8 OPTIONAL,
  bitmapCacheID           Integer8 OPTIONAL,
  destLeft                Coordinate OPTIONAL,
  destTop                 Coordinate OPTIONAL,
  destWidth               Coordinate OPTIONAL,
  destHeight              Coordinate OPTIONAL,
  rop3                    ROP3 OPTIONAL,
  sourceX                 Coordinate OPTIONAL,
  sourceY                 Coordinate OPTIONAL,
  backgroundColor         Color OPTIONAL,
  foregroundColor         Color OPTIONAL,
  brush                   Brush OPTIONAL,
  bitmapCacheIndex        Integer16 OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

TextOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  backMixMode             BackgroundMixMode OPTIONAL,
  startX                  Coordinate OPTIONAL,
  startY                  Coordinate OPTIONAL,
  backgroundColor         Color OPTIONAL,
  foregroundColor         Color OPTIONAL,
  extraSpacing            Integer16 OPTIONAL,
  totalBreakSpacing       Integer16 OPTIONAL,
  breakCount              Integer16 OPTIONAL,
  fontHeight              Integer16 OPTIONAL,
  fontWidth               Integer16 OPTIONAL,
  fontWeight              Integer16 OPTIONAL,
  textFlags               TextAttributeFlags OPTIONAL,
  fontID                  Integer16 OPTIONAL,
  codePointList           ASString(SIZE (1..255)) OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
```

```
  -- Subject to capability negotiation.
  ...
}

ExtendedTextOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  backMixMode             BackgroundMixMode OPTIONAL,
  startX                  Coordinate OPTIONAL,
  startY                  Coordinate OPTIONAL,
  backgroundColor         Color OPTIONAL,
  foregroundColor         Color OPTIONAL,
  extraSpacing            Integer16 OPTIONAL,
  totalBreakSpacing       Integer16 OPTIONAL,
  breakCount              Integer16 OPTIONAL,
  fontHeight              Integer16 OPTIONAL,
  fontWidth               Integer16 OPTIONAL,
  fontWeight              Integer16 OPTIONAL,
  textFlags1              TextAttributeFlags OPTIONAL,
  fontID                  Integer16 OPTIONAL,
  textFlags2              ExtraTextFlags OPTIONAL,
  clipLeft                Coordinate OPTIONAL,
  clipTop                 Coordinate OPTIONAL,
  clipRight               Coordinate OPTIONAL,
  clipBottom              Coordinate OPTIONAL,
  codePointList           ASString(SIZE (1..255)) OPTIONAL,
  deltaXList              SEQUENCE (SIZE (1..127)) OF Coordinate OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FrameOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  destLeft                Coordinate OPTIONAL,
  destTop                 Coordinate OPTIONAL,
  destWidth               Coordinate OPTIONAL,
  destHeight              Coordinate OPTIONAL,
  rop3                    ROP3 OPTIONAL,
  backgroundColor         Color OPTIONAL,
  foregroundColor         Color OPTIONAL,
  brush                   Brush OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

RectangleOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  backMixMode             BackgroundMixMode OPTIONAL,
  destLeft                Coordinate OPTIONAL,
  destTop                 Coordinate OPTIONAL,
  destRight               Coordinate OPTIONAL,
  destBottom              Coordinate OPTIONAL,
  backgroundColor         Color OPTIONAL,
  foregroundColor         Color OPTIONAL,
  brush                   Brush OPTIONAL,
  rop2                    ROP2 OPTIONAL,
  pen                     Pen OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

OpaqueRectangleOrder ::= SEQUENCE {
```

```
  header                  PrimaryOrderHeader,
  destLeft                Coordinate OPTIONAL,
  destTop                 Coordinate OPTIONAL,
  destWidth               Coordinate OPTIONAL,
  destHeight              Coordinate OPTIONAL,
  color                   Color OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

LineOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  backMixMode             BackgroundMixMode OPTIONAL,
  startX                  Coordinate OPTIONAL,
  startY                  Coordinate OPTIONAL,
  endX                    Coordinate OPTIONAL,
  endY                    Coordinate OPTIONAL,
  backgroundColor         Color OPTIONAL,
  rop2                    ROP2 OPTIONAL,
  pen                     Pen OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DesktopSaveOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  saveOffset              Integer32 OPTIONAL,
  destLeft                Coordinate OPTIONAL,
  destTop                 Coordinate OPTIONAL,
  destWidth               Coordinate OPTIONAL,
  destHeight              Coordinate OPTIONAL,
  action                  DesktopSaveAction OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

DesktopOriginOrder ::= SEQUENCE {
  header                  PrimaryOrderHeader,
  desktopLeft             Coordinate OPTIONAL,
  desktopTop              Coordinate OPTIONAL,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ColorSpaceOrder ::= SEQUENCE {
  colorSpace              ColorSpaceSpecifier,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

PrimaryOrder ::= CHOICE {
  destinationBlt            [0]  DestinationBltOrder,
  patternBlt                [1]  PatternBltOrder,
  screenBlt                 [2]  ScreenBltOrder,
  memoryBlt                 [13]  MemoryBltOrder,
  memoryThreeWayBlt         [14]  MemoryThreeWayBltOrder,
  text                      [5]  TextOrder,
  extendedText              [6]  ExtendedTextOrder,
  frame                     [9]  FrameOrder,
```

```
  rectangle                [7]  RectangleOrder,
  line                     [8]  LineOrder,
  opaqueRectangle          [10]  OpaqueRectangleOrder,
  desktopSave              [11]  DesktopSaveOrder,
  desktopOrigin            [32]  DesktopOriginOrder,
  nonStandardPrimaryOrder  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

SecondaryOrder ::= CHOICE {
  cacheBitmap                 [0]  CacheBitmapOrder,
  cacheColorTable             [1]  CacheColorTableOrder,
  colorSpaceOrder             [2]  ColorSpaceOrder,
  nonStandardSecondaryOrder   [3]  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

UpdateOrder ::= CHOICE {
  primaryOrder      PrimaryOrder,
  secondaryOrder    SecondaryOrder,
  nonStandardOrder  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

--|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
--
--           Begin AS PDU Definitions
--
--|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

ApplicationPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  action
    CHOICE {notifyHostedApplications  [1]  NotifyHostedApplications,
            unhostApplication         [2]  UnhostApplication,
            nonStandardAction         NonStandardParameter,
            -- Subject to capability negotiation.
            ...},
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ControlPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  action
    CHOICE {requestControl     [1]  RequestControl,
            grantControl       [2]  GrantControl,
            detach             [3]  Detach,
            cooperate          [4]  Cooperate,
            nonStandardAction  NonStandardParameter,
            -- Subject to capability negotiation.
            ...},
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FlowResponsePDU ::= SEQUENCE {
  flowIdentifier         INTEGER(0..127),
  flowNumber             Integer8,
```

```
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FlowStopPDU ::= SEQUENCE {
  flowIdentifier         INTEGER(0..127),
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FlowTestPDU ::= SEQUENCE {
  flowIdentifier         INTEGER(0..127),
  flowNumber             Integer8,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

FontPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  fontList               SEQUENCE OF FontAttribute,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

InputPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  eventList              SEQUENCE OF InputEvent,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

MediatedControlPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  action
    CHOICE {takeControlRequest        [1]  TakeControlRequest,
            passControlRequest        [2]  PassControlRequest,
            detachRequest             [3]  DetachRequest,
            confirmTakeResponse       [5]  ConfirmTakeResponse,
            denyTakeResponse          [6]  DenyTakeResponse,
            confirmDetachResponse     [7]  ConfirmDetachResponse,
            denyDetachResponse        [8]  DenyDetachResponse,
            denyPassResponse          [9]  DenyPassResponse,
            remoteDetachRequest       [10]  RemoteDetachRequest,
            denyRemoteDetachResponse  [11]  DenyRemoteDetachResponse,
            nonStandardAction         NonStandardParameter,
            -- Subject to capability negotiation.
            ...},
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

PointerPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  pointerData
    CHOICE {systemPointerType      [1]  SystemPointerType,
            monoPointerAttribute   [2]  MonoPointerAttribute,
            colorPointerAttribute  [6]  ColorPointerAttribute,
            cachedPointerIndex     [7]  Integer16,
```

```
            pointerPosition         [3]  Point16,
            nonStandardPointer      NonStandardParameter,
            -- Subject to capability negotiation.
            ...},
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

RemoteSharePDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  action
    CHOICE {requestRemoteShare  [1]  RequestRemoteShare,
            confirmRemoteShare  [2]  ConfirmRemoteShare,
            denyRemoteShare     [3]  DenyRemoteShare,
            nonStandardAction   NonStandardParameter,
            -- Subject to capability negotiation.
            ...},
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

SynchronizePDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  targetUser             UserID,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

UpdateBitmapPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  destLeft               Coordinate16,
  destTop                Coordinate16,
  destRight              Coordinate16,
  destBottom             Coordinate16,
  width                  Integer16,
  height                 Integer16,
  bitsPerPixel           INTEGER(1 | 4 | 8),
  compressedFlag         BOOLEAN,
  bitmapData             BitmapData,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

UpdateOrdersPDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  orderList              SEQUENCE OF UpdateOrder,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

UpdatePalettePDU ::= SEQUENCE {
  shareDataHeader        ShareDataHeader,
  palette                ColorPalette,
  nonStandardParameters  SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

UpdateSynchronizePDU ::= SEQUENCE {
```

```
  shareDataHeader         ShareDataHeader,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

WindowActivationPDU ::= SEQUENCE {
  shareDataHeader         ShareDataHeader,
  action
    CHOICE {localWindowActive           [1]   LocalWindowActiveIndication,
            hostedWindowActive          [2]   HostedWindowActiveIndication,
            hostedWindowInvisible       [3]   HostedWindowInvisibleIndication,
            pointerDeviceCapture        [4]   PointerDeviceCaptureIndication,
            activateWindow              [12769]  ActivateWindowRequest,
            closeWindow                 [12770]  CloseWindowRequest,
            restoreWindow               [12771]  RestoreWindowRequest,
            windowManagerMenu           [12772]  WindowManagerMenuRequest,
            activationHelpKey           [12785]  ActivationHelpKeyRequest,
            activationHelpIndexKey      [12786]  ActivationHelpIndexKeyRequest,
            activationHelpExtendedKey   [12787] ActivationHelpExtendedKeyRequest,
            nonStandardAction           NonStandardParameter,
            -- Subject to capability negotiation.
            ...},
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

WindowListPDU ::= SEQUENCE {
  shareDataHeader         ShareDataHeader,
  listTime                Integer16,
  listID                  Integer16,
  windowAttributeList     SEQUENCE OF WindowAttribute,
  windowTitleList         SEQUENCE OF WindowTitle,
  nonStandardParameters   SEQUENCE OF NonStandardParameter OPTIONAL,
  -- Subject to capability negotiation.
  ...
}

ASNonStandardPDU ::= SEQUENCE {
  nonStandardParameter  NonStandardParameter,
  -- Subject to capability negotiation.
  ...
}

SharePDU ::= CHOICE {
  applicationPDU          [25]  ApplicationPDU,
  controlPDU              [20]  ControlPDU,
  flowResponsePDU         [66]  FlowResponsePDU,
  flowStopPDU             [67]  FlowStopPDU,
  flowTestPDU             [65]  FlowTestPDU,
  fontPDU                 [11]  FontPDU,
  inputPDU                [28]  InputPDU,
  mediatedControlPDU      [29]  MediatedControlPDU,
  pointerPDU              [27]  PointerPDU,
  remoteSharePDU          [30]  RemoteSharePDU,
  synchronizePDU          [31]  SynchronizePDU,
  updateBitmapPDU         [1]  UpdateBitmapPDU,
  updateOrdersPDU         [0]  UpdateOrdersPDU,
  updateSynchronizePDU    [3]  UpdateSynchronizePDU,
  updatePalettePDU        [2]  UpdatePalettePDU,
  windowActivationPDU     [23]  WindowActivationPDU,
  windowListPDU           [24]  WindowListPDU,
  asNonStandardPDU       ASNonStandardPDU,
```

```
  ...
}

--/////////////////////////////////////////////////////////////////
--
--                          End AS Definitions
--
--/////////////////////////////////////////////////////////////////
END
```

9.3 Legacy mode encoding rules

The AS encoding for ASPDU data elements defined in clause 9.1 is as follows.

The bits of the bit combinations of an octet are identified by b_7, b_6, b_5, b_4, b_3, b_2, b_1 and b_0, where b_7 is the highest-order, or most significant, bit and b_0 is the lowest-order, or least significant, bit.

- An octet within a data element is encoded as a sequence of bits, where b_7 of the octet is encoded into the highest-order, or most significant, bit of the corresponding encoded octet, b_6 is encoded into the next highest-order bit, and so on, filling towards the least significant bit.
- OCTET STRING is encoded as a sequence of octets in the order in which they appear in the data element.
- INTEGER (0..15) is encoded to the highest-order, or most significant, available four bits within an octet.
- INTEGER (0..255) and INTEGER (–128..127) are encoded to an octet containing the two's complement binary value of the data element.
- INTEGER (0..4095) is encoded to the highest-order, or most significant, available twelve bits within an octet.
- INTEGER (0..65535) and INTEGER (–32768..32767) are encoded to two octets containing the two's complement binary value of the data element, where the highest-order, or most significant, octet is placed in the second octet.
- INTEGER (0..4294967295) is encoded to four octets containing the two's complement binary value of the data element; octets are arranged in increasing significance, where the highest-order, or most significant, octet is placed in the fourth octet.
- BIT STRING (0..7) is encoded as a single octet.
- BIT STRING (0..15) is encoded as INTEGER (0..65535).
- BIT STRING (0..31) is encoded as INTEGER (0..4294967295).
- All bits are packed to bit boundaries. Pad elements are explicitly defined.
- Where a data element is a member of a CHOICE type, the specific data element encoded for the choice depends on other data elements as described in the protocol specification (see clause 8). No additional bits are encoded for such data elements.
- Where a data element is OPTIONAL, whether the data element is encoded or not depends on other data elements as described in the protocol specification (see clause 8). No additional bits are encoded for such data elements.
- Where the usage of a data element is not specified in the protocol specification (clause 8), the data element is encoded as above, but the bit values are undefined.

9.4 Base mode non-collapsing capabilities encoding rules

In the base mode of the AS protocol, an ASCE may advertise a particular capability via either the collapsing or non-collapsing capabilities lists in the roster (see clause 8.2.2). Where an ASCE

advertises a particular capability via the non-collapsing capabilities list in the roster, then it shall encode the capability value using the encoding rules defined in this clause.

In the base mode of the AS protocol, capabilities may be one of the classes defined in Table 8-1.

The defined capability classes are encoded using the BASIC ALIGNED variant of the packed encoding rules of [ITU-T X.691] based on the following ASN.1 definitions.

```
LogicalNonCollapsingCapability ::= BOOLEAN                -- Class L: logical value.
IntegerNonCollapsingCapability ::= INTEGER (MIN..MAX)     -- Class N: signed or unsigned integer value.
```

After encoding, the encoded value for a particular capability is used as the non-collapsing capability value for subsequent encoding by GCC.

Annex A

Static channel ID assignments

(This annex forms an integral part of this Recommendation)

Table A.1 lists the numerical assignment of static channel IDs for the static channels allocated for use by this Recommendation. The numerical assignment of static channel IDs is listed in [ITU-T T.120].

Table A.1 – Static channel ID assignments

Symbolic name	Channel ID
AS-CHANNEL-0	11

Annex B

Legacy application protocol key

(This annex forms an integral part of this Recommendation)

Table B.1 defines the contents of the application protocol key used to identify the legacy mode of the application protocol defined by this Recommendation.

Table B.1 – Legacy application protocol key

Application protocol key	Description
h221NonStandard: 0xB5, 0x00, 0x53, 0x4C, 0x02	This defines the contents of the application protocol key used to identify the legacy mode of the application protocol defined by this Recommendation. Note that the H.221 non-standard identifier option is used to define this key. The numerical values shown represent the contents of the H221NonStandardIdentifier as defined in [ITU-T T.124].

Annex C

Object identifier assignments

(This annex forms an integral part of this Recommendation)

Table C.1 lists the assignment of object identifiers defined for use by this Recommendation.

Table C.1

Object identifier value	Description
{itu-t recommendation t 128 version (0) 1}	This object identifier is used to indicate the version of this Recommendation.

Appendix I

Informative values

(This appendix does not form an integral part of this Recommendation)

This appendix provides suggestions for various values described within the main body of this Recommendation, based on experience with application sharing on a number of terminal types. These values are not mandatory and the actual values used by a specific ASCE are left to the discretion of the implementer.

I.1 Flow control

The following values and expressions are suggested for use in the flow control algorithm described in clause 8.5. See Tables I.1, I.2 and I.3.

Table I.1 – Flow control constants

Item	MCS high priority	MCS medium priority	MCS low priority
target_round_trip	2 000	Not flow controlled	7 000
target_in_flight	800	Not flow controlled	99 000
max_queued_recv	5	Not flow controlled	5

Table I.2 – Flow control variables

Item	Initial value	Minimum	Maximum
flow_period (milliseconds)	1 000	100	1 000
max_in_flight	8 000	500	256 000

Table I.3 – Flow control operations

Operation	Expression
Decrease max_in_flight	max_in_flight = max_in_flight/2 (but not below minimum value in Table I.2)
Increase max_in_flight	max_in_flight = max_in_flight*2 (but not above the maximum value in Table I.2)
Decrease flow_period	flow_period = flow_period/2 (but not below minimum value in Table I.2)
Increase flow_period	flow_period = flow_period*2 (but not above maximum value in Table I.2)

I.2 Bitmap caching

The following values are suggested for use in the bitmap cache capability set described in clause 8.2.7.

Item	Suggested value
cache1Entries	600
cache1MaximumCellSize	496
cache2Entries	300
cache2MaximumCellSize	2 032
cache3Entries	150
cache3MaximumCellSize	4 080

I.3 ColorTable caching

The following values are suggested for use in the ColorTable cache capability set described in clause 8.2.8.

Item	Suggested value
colorTableCacheSize	6

I.4 Pointer caching

The following values are suggested for use in the pointer capability set described in clause 8.2.11.

Item	Suggested value
colorPointerFlag	TRUE
pointerCacheSize	25

I.5 Desktop save cache

The following values are suggested for use for the desktop save cache values in the order capability set described in clause 8.2.5 and for the desktop cache algorithm described in clause 8.16.17.

Item	Suggested value
desktopSaveSize	160,000
desktopSaveXGranularity	1
desktopSaveYGranularity	20

I.6 General compression

The following are suggested values for use in legacy mode for negotiation and usage of general compression, as described in clause 8.3.2.

Compression scheme	generalCompressionTypes (see Table 8-3)	generalCompressionLevel (see Table 8-3)	generalCompressedType (see Table 8-23)
PKWARE PKZIP	Bit flag 0 set	0 and 1	1

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	**Terminals for telematic services**
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2009