[MS-RDPEGDI]:
Remote Desktop Protocol:
Graphics Device Interface (GDI) Acceleration Extensions

Revision Summary

Date	Revision History	Revision Class	Comments
02/22/2007	0.01		MCPP Milestone 3 Initial Availability
06/01/2007	1.0	Major	Updated and revised the technical content.
07/03/2007	1.1	Minor	Minor technical content updates.
07/20/2007	1.1.1	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
08/10/2007	1.2	Minor	Updated content based on feedback.
09/28/2007	1.3	Minor	Made technical and editorial changes based on feedback.
10/23/2007	1.3.1	Editorial	Revised and edited the technical content.
11/30/2007	1.4	Minor	Made technical and editorial changes based on feedback.
01/25/2008	2.0	Major	Updated and revised the technical content.

Table of Contents

1 Introduction 10
1.1 Glossary 10
1.2 References 10
1.2.1 Normative References 10
1.2.2 Informative References 11
1.3 Protocol Overview (Synopsis) 12
1.3.1 Accelerated Graphics 12
1.3.1.1 Caches 12
1.3.1.2 Drawing Orders 13
1.3.1.2.1 Primary Drawing Orders 13
1.3.1.2.2 Secondary Drawing Orders 14
1.3.1.2.3 Alternate Secondary Drawing Orders 14
1.3.1.3 Error Conditions 15
1.3.2 RDP 6.0 Bulk Compression 15
1.3.3 Server Redirection 15
1.4 Relationship to Other Protocols 17
1.5 Prerequisites/Preconditions 17
1.6 Applicability Statement 17
1.7 Versioning and Capability Negotiation 17
1.8 Vendor-Extensible Fields 17
1.9 Standards Assignments 17

2 Messages 18
2.1 Transport 18
2.2 Message Syntax 18
2.2.1 Capability Sets 18
2.2.1.1 Color Table Cache Capability Set (TS_COLORTABLE_CAPABILITYSET) 18
2.2.1.2 DrawNineGrid Cache Capability Set (TS_DRAW_NINEGRID_CAPABILITYSET) 19
2.2.1.3 Draw GDI+ Capability Set (TS_DRAW_GDIPLUS_CAPABILITYSET) 20
2.2.1.3.1 GDI+ Cache Entries (TS_GDIPLUS_CACHE_ENTRIES) 21
2.2.1.3.2 GDI+ Cache Chunk Size (TS_GDIPLUS_CACHE_CHUNK_SIZE) 22
2.2.1.3.3 GDI+ Image Cache Properties (TS_GDIPLUS_IMAGE_CACHE_PROPERTIES) 22
2.2.2 Accelerated Graphics 23
2.2.2.1 RDP 6.0 Bitmap Compressed Bitmap Stream (RDP6_BITMAP_STREAM) 23
2.2.2.1.1 RDP 6.0 RLE Segments (RDP6_RLE_SEGMENTS) 25
2.2.2.1.2 RDP 6.0 RLE Segment (RDP6_RLE_SEGMENT) 25
2.2.2.2 Orders Update (TS_UPDATE_ORDERS_PDU_DATA) 26
2.2.2.2.1 Drawing Order (DRAWING_ORDER) 27
2.2.2.3 Fast-Path Orders Update (TS_FP_UPDATE_ORDERS) 27
2.2.2.3.1 Drawing Order (DRAWING_ORDER) 27
2.2.2.3.1.1 Primary Drawing Orders 28
2.2.2.3.1.1.1 Common Data Types 28
2.2.2.3.1.1.1.1 Coord Field (COORD_FIELD) 28
2.2.2.3.1.1.1.2 One-Byte Header Variable Field (VARIABLE1_FIELD) 29
2.2.2.3.1.1.1.3 Two-Byte Header Variable Field (VARIABLE2_FIELD) 29
2.2.2.3.1.1.1.4 Delta-Encoded Points (DELTA_PTS_FIELD) 29
2.2.2.3.1.1.1.5 Delta-Encoded Rectangles (DELTA_RECTS_FIELD) 30
2.2.2.3.1.1.1.6 Binary Raster Operation (ROP2_OPERATION) 32
2.2.2.3.1.1.1.7 Ternary Raster Operation Index (ROP3_OPERATION_INDEX) 33
2.2.2.3.1.1.1.8 Generic Color (TS_COLOR) 42
2.2.2.3.1.1.1.9 Fill Mode (FILL_MODE) 43
2.2.2.3.1.1.2 Primary Drawing Order (PRIMARY_DRAWING_ORDER) 43

2.2.2.3.1.1.2.1 DstBlt (DSTBLT_ORDER) 48
2.2.2.3.1.1.2.2 MultiDstBlt (MULTI_DSTBLT_ORDER) 49
2.2.2.3.1.1.2.3 PatBlt (PATBLT_ORDER) 50
2.2.2.3.1.1.2.4 MultiPatBlt (MULTI_PATBLT_ORDER) 53
2.2.2.3.1.1.2.5 OpaqueRect (OPAQUERECT_ORDER) 55
2.2.2.3.1.1.2.6 MultiOpaqueRect (MULTI_OPAQUERECT_ORDER) 56
2.2.2.3.1.1.2.7 ScrBlt (SCRBLT_ORDER)....58
2.2.2.3.1.1.2.8 MultiScrBlt (MULTI_SCRBLT_ORDER)....60
2.2.2.3.1.1.2.9 MemBlt (MEMBLT_ORDER)....62
2.2.2.3.1.1.2.10 Mem3Blt (MEM3BLT_ORDER) 64
2.2.2.3.1.1.2.11 LineTo (LINETO_ORDER) 66
2.2.2.3.1.1.2.12 SaveBitmap (SAVEBITMAP_ORDER) 68
2.2.2.3.1.1.2.13 GlyphIndex (GLYPHINDEX_ORDER)....70
2.2.2.3.1.1.2.14 FastIndex (FASTINDEX_ORDER) 74
2.2.2.3.1.1.2.15 FastGlyph (FASTGLYPH_ORDER)....76
2.2.2.3.1.1.2.16 PolygonSC (POLYGON_SC_ORDER) 78
2.2.2.3.1.1.2.17 PolygonCB (POLYGON_CB_ORDER) 79
2.2.2.3.1.1.2.18 Polyline (POLYLINE_ORDER) 81
2.2.2.3.1.1.2.19 EllipseSC (ELLIPSE_SC_ORDER) 82
2.2.2.3.1.1.2.20 EllipseCB (ELLIPSE_CB_ORDER) 84
2.2.2.3.1.1.2.21 DrawNineGrid (DRAWNINEGRID_ORDER)....85
2.2.2.3.1.1.2.22 MultiDrawNineGrid (MULTI_DRAWNINEGRID_ORDER)....86
2.2.2.3.1.2 Secondary Drawing Orders 88
2.2.2.3.1.2.1 Common Data Types 88
2.2.2.3.1.2.1.1 Secondary Drawing Order Header (SECONDARY_DRAWING_ORDER_HEADER) 88
2.2.2.3.1.2.1.2 Two-Byte Unsigned Encoding (TWO_BYTE_UNSIGNED_ENCODING)....89
2.2.2.3.1.2.1.3 Two-Byte Signed Encoding (TWO_BYTE_SIGNED_ENCODING) 90
2.2.2.3.1.2.1.4 Four-Byte Unsigned Encoding (FOUR_BYTE_UNSIGNED_ENCODING)....90
2.2.2.3.1.2.2 Cache Bitmap - Revision 1 (CACHE_BITMAP_ORDER)....91
2.2.2.3.1.2.3 Cache Bitmap - Revision 2 (CACHE_BITMAP_REV2_ORDER)....93
2.2.2.3.1.2.4 Cache Color Table (CACHE_COLOR_TABLE_ORDER)....96
2.2.2.3.1.2.4.1 Color Quad (TS_COLOR_QUAD)....97
2.2.2.3.1.2.5 Cache Glyph - Revision 1 (CACHE_GLYPH_ORDER) 97
2.2.2.3.1.2.5.1 Cache Glyph Data (TS_CACHE_GLYPH_DATA)....98
2.2.2.3.1.2.6 Cache Glyph - Revision 2 (CACHE_GLYPH_REV2_ORDER) 99
2.2.2.3.1.2.6.1 Cache Glyph Data - Revision 2 (TS_CACHE_GLYPH_DATA_REV2) 100
2.2.2.3.1.2.7 Cache Brush (CACHE_BRUSH_ORDER)....101
2.2.2.3.1.2.7.1 Compressed Color Brush (COMPRESSED_COLOR_BRUSH) 102
2.2.2.3.1.3 Alternate Secondary Drawing Orders....103
2.2.2.3.1.3.1 Common Data Types 103
2.2.2.3.1.3.1.1 Alternate Secondary Drawing Order Header (ALTSEC_DRAWING_ORDER_HEADER) 103
2.2.2.3.1.3.2 Create Offscreen Bitmap (CREATE_OFFSCR_BITMAP_ORDER) 105
2.2.2.3.1.3.2.1 Offscreen Cache Delete List (OFFSCR_DELETE_LIST) 106
2.2.2.3.1.3.3 Switch Surface (SWITCH_SURFACE_ORDER)....106
2.2.2.3.1.3.4 Create NineGrid Bitmap (CREATE_NINEGRID_BITMAP_ORDER) 106
2.2.2.3.1.3.4.1 NineGrid Bitmap Information (NINEGRID_BITMAP_INFO) 107
2.2.2.3.1.3.4.1.1 Color Reference (TS_COLORREF)....109
2.2.2.3.1.3.5 Stream Bitmap Orders....109
2.2.2.3.1.3.5.1 Stream Bitmap First (STREAM_BITMAP_FIRST_ORDER) 109
2.2.2.3.1.3.5.2 Stream Bitmap Next (STREAM_BITMAP_NEXT_ORDER)....111

2.2.2.3.1.3.6 GDI+ Orders 112
2.2.2.3.1.3.6.1 Common Data Types 112
2.2.2.3.1.3.6.1.1 GDI+ Cache Type (DRAW_GDIPLUS_CACHE_TYPE) 112
2.2.2.3.1.3.6.2 Draw GDI+ Cache First (DRAW_GDIPLUS_CACHE_FIRST_ORDER) 113
2.2.2.3.1.3.6.3 Draw GDI+ Cache Next (DRAW_GDIPLUS_CACHE_NEXT_ORDER) 113
2.2.2.3.1.3.6.4 Draw GDI+ Cache End (DRAW_GDIPLUS_CACHE_END_ORDER) 114
2.2.2.3.1.3.6.5 Draw GDI+ First (DRAW_GDIPLUS_FIRST_ORDER) 115
2.2.2.3.1.3.6.6 Draw GDI+ Next (DRAW_GDIPLUS_NEXT_ORDER) 116
2.2.2.3.1.3.6.7 Draw GDI+ End (DRAW_GDIPLUS_END_ORDER) 117
2.2.2.4 Error Conditions 118
2.2.2.4.1 Client Bitmap Cache Error PDU 118
2.2.2.4.1.1 Bitmap Cache Error PDU Data (TS_BITMAP_CACHE_ERROR_PDU) 119
2.2.2.4.1.1.1 Bitmap Cache Error Info (TS_BITMAP_CACHE_ERROR_INFO) 120
2.2.2.4.2 Client Offscreen Bitmap Cache Error PDU 120
2.2.2.4.2.1 Offscreen Bitmap Cache Error PDU Data (TS_OFFSCRCACHE_ERROR_PDU) 122
2.2.2.4.3 Client DrawNineGrid Cache Error PDU 122
2.2.2.4.3.1 DrawNineGrid Cache Error PDU Data (TS_DRAWNINEGRID_ERROR_PDU) 124
2.2.2.4.4 Client GDI+ Error PDU 124
2.2.2.4.4.1 GDI+ Cache Error PDU Data (TS_DRAWGDIPLUS_ERROR_PDU) 126
2.2.3 Server Redirection 126
2.2.3.1 Server Redirection Packet (RDP_SERVER_REDIRECTION_PACKET) 126
2.2.3.1.1 Target Net Addresses (TARGET_NET_ADDRESSES) 130
2.2.3.1.1.1 Target Net Address (TARGET_NET_ADDRESS) 130
2.2.3.2 Standard RDP Security 130
2.2.3.2.1 Standard Security Server Redirection PDU 130
2.2.3.3 Enhanced RDP Security 132
2.2.3.3.1 Enhanced Security Server Redirection PDU 132

3 Protocol Details 134
3.1 Common Details 134
3.1.1 Abstract Data Model 134
3.1.1.1 Caches 134
3.1.1.1.1 Bitmap Caches 134
3.1.1.1.2 Glyph and Fragment Caches 135
3.1.1.1.3 Color Table Cache 136
3.1.1.1.4 Brush Caches 136
3.1.1.1.5 Offscreen Bitmap Cache 136
3.1.1.1.6 NineGrid Bitmap Cache 136
3.1.1.1.7 GDI+ Caches 137
3.1.2 Timers 137
3.1.3 Initialization 137
3.1.4 Higher-Layer Triggered Events 137
3.1.5 Message Processing Events and Sequencing Rules 137
3.1.6 Timer Events 137
3.1.7 Other Local Events 137
3.1.8 RDP 6.0-Based Bulk Data Compression 137
3.1.8.1 Abstract Data Model 137
3.1.8.2 Compressing Data 138
3.1.8.3 Decompressing Data 138
3.1.8.4 Wire Format 138
3.1.8.4.1 Literal, EOS, and Copy-Offset Tables 139

3.1.8.4.2 Length-of-Match Tables 143
3.1.8.4.3 Encoding the Logically Compressed Stream 145
3.1.8.4.3.1 Encoding the Copy-Offset 147
3.1.8.4.3.1.1 Examples of Copy-Offset Encoding 147
3.1.8.4.3.2 Encoding the Length-of-Match 147
3.1.8.4.3.2.1 Examples of Length-of-Match Encoding 148
3.1.8.4.4 Decoding a Compressed Stream 148
3.1.8.4.4.1 Decoding the Copy-Offset 150
3.1.8.4.4.1.1 Examples of Copy-Offset Decoding 150
3.1.8.4.4.2 Decoding the Length-of-Match 150
3.1.8.4.4.2.1 Examples of Length-of-Match Decoding 151
3.1.9 RDP 6.0 Bitmap Compression 151
3.1.9.1 Bitmap Compression Techniques 151
3.1.9.1.1 Splitting and Combining Color Planes 151
3.1.9.1.2 Color Space Conversion 152
3.1.9.1.3 Chroma Subsampling and Super-Sampling 153
3.1.9.1.4 Color Loss Reduction 154
3.1.9.2 Run-Length Encoding 155
3.1.9.2.1 Encoding Run-Length Sequences 156
3.1.9.2.2 Extra Long RUN Sequences 158
3.1.9.2.3 Decoding Run-Length Sequences 159
3.1.9.3 Compressing a Bitmap 161
3.1.9.4 Decompressing a Bitmap 162
3.2 Client Details 163
3.2.1 Abstract Data Model 163
3.2.1.1 Primary Drawing Order History 164
3.2.1.2 Save Bitmap 164
3.2.2 Timers 164
3.2.3 Initialization 164
3.2.4 Higher-Layer Triggered Events 164
3.2.5 Message Processing Events and Sequencing Rules 164
3.2.5.1 Drawing Orders 164
3.2.5.1.1 Primary Drawing Orders 165
3.2.5.1.1.1 Processing Primary Drawing Orders 165
3.2.5.1.1.1.1 Processing of DstBlt 166
3.2.5.1.1.1.2 Processing of MultiDstBlt 166
3.2.5.1.1.1.3 Processing of PatBlt 166
3.2.5.1.1.1.4 Processing of MultiPatBlt 166
3.2.5.1.1.1.5 Processing of OpaqueRect 166
3.2.5.1.1.1.6 Processing of MultiOpaqueRect 166
3.2.5.1.1.1.7 Processing of ScrBlt 167
3.2.5.1.1.1.8 Processing of MultiScrBlt 167
3.2.5.1.1.1.9 Processing of MemBlt 167
3.2.5.1.1.1.10 Processing of Mem3Blt 167
3.2.5.1.1.1.11 Processing of LineTo 168
3.2.5.1.1.1.12 Processing of SaveBitmap 168
3.2.5.1.1.1.13 Processing of GlyphIndex 168
3.2.5.1.1.1.14 Processing of FastIndex 168
3.2.5.1.1.1.15 Processing of FastGlyph 168
3.2.5.1.1.1.16 Processing of PolygonSC 169
3.2.5.1.1.1.17 Processing of PolygonCB 169
3.2.5.1.1.1.18 Processing of Polyline 169
3.2.5.1.1.1.19 Processing of EllipseSC 169
3.2.5.1.1.1.20 Processing of EllipseCB 169
3.2.5.1.1.1.21 Processing of DrawNineGrid 169

3.2.5.1.1.1.22 Processing of MultiDrawNineGrid 170
3.2.5.1.2 Secondary Drawing Orders 170
3.2.5.1.2.1 Processing Secondary Drawing Orders 170
3.2.5.1.2.1.1 Processing of Cache Bitmap (Revision 1) 170
3.2.5.1.2.1.2 Processing of Cache Bitmap (Revision 2) 171
3.2.5.1.2.1.3 Processing of Cache Color Table 171
3.2.5.1.2.1.4 Processing of Cache Glyph (Revision 1) 171
3.2.5.1.2.1.5 Processing of Cache Glyph (Revision 2) 171
3.2.5.1.2.1.6 Processing of Cache Brush 171
3.2.5.1.3 Alternate Secondary Drawing Orders 172
3.2.5.1.3.1 Processing Alternate Secondary Drawing Orders 172
3.2.5.1.3.1.1 Processing of Create Offscreen Bitmap 172
3.2.5.1.3.1.2 Processing of Switch Surface 172
3.2.5.1.3.1.3 Processing of Create NineGrid Bitmap 173
3.2.5.1.3.1.4 Processing of Stream Bitmap Orders 173
3.2.5.1.3.1.5 GDI+ Orders 173
3.2.5.1.3.1.5.1 Processing of Draw GDI+ Cache Orders 173
3.2.5.1.3.1.5.2 Processing of Draw GDI+ Orders 173
3.2.5.2 Error Conditions 174
3.2.5.2.1 Sending of Bitmap Cache Error PDU 174
3.2.5.2.2 Sending of the Offscreen Bitmap Cache Error PDU 174
3.2.5.2.3 Sending of the DrawNineGrid Cache Error PDU 174
3.2.5.2.4 Sending of the GDI+ Error PDU 175
3.2.5.3 Processing of the Server Redirection PDU 175
3.2.6 Timer Events 175
3.2.7 Other Local Events 175
3.3 Server Details 176
3.3.1 Abstract Data Model 176
3.3.1.1 Persistent Bitmap Keys 176
3.3.1.2 Primary Drawing Order History 176
3.3.1.3 Bitmap Cache Wait List 176
3.3.2 Timers 176
3.3.3 Initialization 177
3.3.4 Higher-Layer Triggered Events 177
3.3.5 Message Processing Events and Sequencing Rules 177
3.3.5.1 Drawing Orders 177
3.3.5.1.1 Primary Drawing Orders 177
3.3.5.1.1.1 Construction of a Primary Drawing Order 177
3.3.5.1.1.1.1 Construction of DstBlt 178
3.3.5.1.1.1.2 Construction of MultiDstBlt 178
3.3.5.1.1.1.3 Construction of PatBlt 178
3.3.5.1.1.1.4 Construction of MultiPatBlt 178
3.3.5.1.1.1.5 Construction of OpaqueRect 179
3.3.5.1.1.1.6 Construction of MultiOpaqueRect 179
3.3.5.1.1.1.7 Construction of ScrBlt 179
3.3.5.1.1.1.8 Construction of MultiScrBlt 179
3.3.5.1.1.1.9 Construction of MemBlt 179
3.3.5.1.1.1.10 Construction of Mem3Blt 180
3.3.5.1.1.1.11 Construction of LineTo 180
3.3.5.1.1.1.12 Construction of SaveBitmap 181
3.3.5.1.1.1.13 Construction of GlyphIndex 181
3.3.5.1.1.1.14 Construction of FastIndex 181
3.3.5.1.1.1.15 Construction of FastGlyph 181
3.3.5.1.1.1.16 Construction of PolygonSC 182
3.3.5.1.1.1.17 Construction of PolygonCB 182

3.3.5.1.1.1.18 Construction of PolyLine 182
3.3.5.1.1.1.19 Construction of EllipseSC 182
3.3.5.1.1.1.20 Construction of EllipseCB 182
3.3.5.1.1.1.21 Construction of DrawNineGrid 183
3.3.5.1.1.1.22 Construction of MultiDrawNineGrid 183
3.3.5.1.2 Secondary Drawing Orders 183
3.3.5.1.2.1 Construction of Secondary Drawing Orders 183
3.3.5.1.2.1.1 Construction of Cache Bitmap (Revision 1) 183
3.3.5.1.2.1.2 Construction of Cache Bitmap (Revision 2) 184
3.3.5.1.2.1.3 Construction of Cache Color Table 184
3.3.5.1.2.1.4 Construction of Cache Glyph (Revision 1) 185
3.3.5.1.2.1.5 Construction of Cache Glyph (Revision 2) 185
3.3.5.1.2.1.6 Construction of Cache Brush 185
3.3.5.1.3 Alternate Secondary Drawing Orders 186
3.3.5.1.3.1 Construction of Alternate Secondary Drawing Orders 186
3.3.5.1.3.1.1 Construction of Create Offscreen Bitmap 186
3.3.5.1.3.1.2 Construction of Switch Surface 186
3.3.5.1.3.1.3 Construction of Create NineGrid Bitmap 187
3.3.5.1.3.1.4 Construction of Stream Bitmap Orders 187
3.3.5.1.3.1.5 GDI+ Orders 187
3.3.5.1.3.1.5.1 Construction of Draw GDI+ Cache Orders 187
3.3.5.1.3.1.5.2 Construction of Draw GDI+ Orders 188
3.3.5.2 Error Conditions 188
3.3.5.2.1 Processing of Bitmap Cache Error PDU 188
3.3.5.2.2 Processing of the Offscreen Bitmap Cache Error PDU 188
3.3.5.2.3 Processing of the DrawNineGrid Cache Error PDU 188
3.3.5.2.4 Processing of the GDI+ Error PDU 188
3.3.5.3 Sending of the Server Redirection PDUs 189
3.3.6 Timer Events 189
3.3.7 Other Local Events 189

4 Protocol Examples 190
4.1 Annotated Primary Drawing Orders 190
4.1.1 DstBlt 190
4.1.2 MultiDstBlt 190
4.1.3 PatBlt 191
4.1.4 MultiPatBlt 192
4.1.5 OpaqueRect 193
4.1.6 MultiOpaqueRect 194
4.1.7 ScrBlt 195
4.1.8 MultiScrBlt 195
4.1.9 MemBlt 198
4.1.10 Mem3Blt 199
4.1.11 LineTo 200
4.1.12 SaveBitmap 201
4.1.13 FastIndex 201
4.1.13.1 Example 1 201
4.1.13.2 Example 2 202
4.1.14 FastGlyph 203
4.1.15 PolygonSC 205
4.1.16 PolygonCB 208
4.1.17 Polyline 209
4.1.18 EllipseSC 211
4.1.19 EllipseCB 212
4.1.20 DrawNineGrid 213

4.1.21 MultiDrawNineGrid 214
4.2 Annotated Secondary Drawing Orders 215
4.2.1 Cache Bitmap (Revision 2) 215
4.2.2 Cache Color Table 218
4.2.3 Cache Glyph (Revision 2) 223
4.2.4 Cache Brush 224
4.3 Annotated Alternate Secondary Drawing Orders 225
4.3.1 Create Offscreen Bitmap 225
4.3.2 Switch Surface 225
4.3.3 Create NineGrid Bitmap 225
4.3.4 Stream Bitmap First 226
4.3.5 Stream Bitmap Next 227
4.4 Standard Security Server Redirection PDU 227
4.5 Enhanced Security Server Redirection PDU 231
4.6 NineGrid Examples 233
4.7 Save Bitmap Example 234
4.8 Glyph Image Data 236
4.8.1 "d" Character 236
4.8.2 "p" Character 237

5 Security 238
5.1 Security Considerations for Implementers 238
5.2 Index of Security Parameters 238

6 Appendix A: Windows Behavior 239

7 Index 240

1 Introduction

The Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extension is an extension to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]). The aim of Remote Desktop Protocol: GDI Acceleration Extension is to reduce the bandwidth associated with graphics remoting by encoding the drawing operations that produce an image instead of encoding the actual image.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Client
Protocol Data Unit (PDU)
Server

The following terms are specific to this document:

Brush: An 8x8 pixel bitmap that is repeated horizontally and vertically to fill an area.

Reverse Polish Notation: A mathematical notation wherein every operator follows all of its operands. Also known as Postfix notation.

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

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-EMFPLUS] Microsoft Corporation, "Enhanced Metafile Format Plus Extensions Specification", June 2007.

[MS-RDPBCGR] Microsoft Corporation, "Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification", June 2007.

[MS-RDPERP] Microsoft Corporation, "Remote Desktop Protocol: Remote Programs Virtual Channel Extension", July 2007.

[PW] Petzold, C., "Programming Windows, Fifth Edition", Microsoft Press, 1998, ISBN: 157231995X.

If you have any trouble finding [PW], please check here.

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

[T123] ITU-T, "Network-Specific Data Protocol Stacks for Multimedia Conferencing", Recommendation T.123, May 1999, http://www.itu.int/rec/T-REC-T.123/en

Note There is a charge to download the specification.

[T125] ITU-T, "Multipoint Communication Service Protocol Specification", Recommendation T.125, February 1998, http://www.itu.int/rec/T-REC-T.125-199802-I/en

Note There is a charge to download the specification.

[T128] ITU-T, "Multipoint Application Sharing", Recommendation T.128, February 1998, http://www.itu.int/rec/T-REC-T.128-199802-I/en

Note There is a charge to download the specification.

[WGFX] Yuan, F., "Windows Graphics Programming - Win32 GDI and DirectDraw", Prentice Hall PTR, 2000, ISBN: 0130869856.

If you have any trouble finding [WGFX], please check here.

[X224] ITU-T, "Information technology - Open Systems Interconnection - Protocol for Providing the Connection-Mode Transport Service", Recommendation X.224, November 1995, http://www.itu.int/rec/T-REC-X.224-199511-I/en

Note There is a charge to download the specification.

1.2.2 Informative References

[CANONHUFF] Sayood, K., "Lossless Compression Handbook, First Edition", Academic Press, August 2002, ISBN: 0126208611.

[HUFFCODE] Usher, M. J. and Guy, C. G., "Information and Communication for Engineers, Illustrate Edition", MacMillan Pub. Ltd., December 1997, ISBN: 0333615271.

[MSDN-ABLEND] Microsoft Corporation, "AlphaBlend", http://msdn2.microsoft.com/en-us/library/ms532324.aspx

[MSDN-BitBlt] Microsoft Corporation, "BitBlt", http://msdn2.microsoft.com/en-us/library/ms532278.aspx

[MSDN-BRO] Microsoft Corporation, "Binary Roster Operations", http://msdn2.microsoft.com/en-us/library/ms534907.aspx

[MSDN-FT] Microsoft Corporation, "Formatting Text", http://msdn2.microsoft.com/en-us/library/ms533997.aspx

[MSDN-TransparentBlt] Microsoft Corporation, "TransparentBlt", http://msdn2.microsoft.com/en-us/library/ms532303.aspx

[MSDN-TRO] Microsoft Corporation, "Ternary Raster Operations", http://msdn2.microsoft.com/en-us/library/ms534885.aspx

[MSFT-SDLBTS] Microsoft Corporation, "Session Directory and Load Balancing Using Terminal Server", September 2002, http://www.microsoft.com/windowsserver2003/techinfo/overview/sessiondirectory.mspx

[NINEGRID] Microsoft Corporation, "Using Nine-Grid Rendering", http://msdn2.microsoft.com/en-us/library/bb189722.aspx

1.3 Protocol Overview (Synopsis)

The Remote Desktop Protocol: GDI Acceleration Extension reduces the bandwidth associated with graphics remoting. The following sections provide an overview of the major components of this protocol and how bandwidth reduction is achieved.

1.3.1 Accelerated Graphics

The remoting of graphics images (see [MS-RDPBCGR] sections 2.2.9.1.1.3.1.2 and 2.2.9.1.2.1.2) is accomplished by continuously sending updated bitmap images from **server** to **client**. Even though these bitmaps may be compressed, it is still not a bandwidth-efficient mechanism to employ, especially when dealing with graphics-intensive applications that refresh regularly.

The Remote Desktop Protocol: GDI Acceleration Extension aims to reduce the bandwidth associated with graphics remoting by encoding the drawing operations that produce an image instead of encoding the actual image.

For example, instead of sending the bitmap image of a filled rectangle from server to client, an order to render a rectangle at coordinate (X, Y) with a given width, height, and fill color is sent to the client. The client then executes the drawing order to produce the intended graphics result.

In addition to defining how to encode common drawing operations, the Remote Desktop Protocol: GDI Acceleration Extension also facilitates the use of caches to store drawing primitives such as bitmaps, color tables, and characters. The effective use of caching techniques helps to reduce wire traffic by ensuring that items used in multiple drawing operations are sent only once from server to client (retransmission of these items for use in conjunction with future drawing operations is not required after the item has been cached on the client).

1.3.1.1 Caches

The Remote Desktop Protocol: GDI Acceleration Extension defines a number of caches that may be leveraged by clients and servers.

- Bitmap Cache: Stores bitmap images.
- Color Table Cache: Stores color palettes.
- Glyph Cache: Stores character images.
- Fragment Cache: Stores collections of glyphs.
- Brush Cache: Stores 8-pixel by 8-pixel bitmaps used to fill regions.
- Offscreen Bitmap Cache: Stores writable bitmaps.
- GDI+ Caches: Used to cache GDI+ 1.1 primitives:
  - Graphics Cache
  - Brush Cache
  - Pen Cache
  - Image Attributes Cache
  - Image Cache

- NineGrid Bitmap Cache: Stores NineGrid-compliant bitmaps (for more information, see [NINEGRID]).

All of the caches (except the bitmap cache) are memory-based and are not persisted across connections. The use of caches is optional and is negotiated through the use of capability sets.

Encoded drawing operations that use cached items refer to these items by specifying the cache entry in which the item is stored (if there are multiple caches, the cache ID also needs to be specified). This implies that an item must first be cached (by the server and client) before any drawing operations that reference it can be sent.

For example, a server may instruct a client to cache a particular **brush** pattern in the Brush Cache, for example, at index location 23. Then, in a subsequent drawing operation that involves a brush pattern, the server may instruct the client to use the brush pattern stored in the Brush Cache at index location 23.

Entries can be evicted from a cache if room needs to be made for new items. The server decides what entries should be evicted, and then instructs the client to perform the eviction. In this way, the client and server caches remain in sync.

1.3.1.2 Drawing Orders

Drawing orders are used to encode the operations necessary to produce a graphic image or to manipulate a particular cache. There are three classes of drawing orders.

- Primary
- Secondary
- Alternate Secondary

Primary drawing orders are generally used to encode drawing operations, while secondary drawing orders are used to manage caches (the addition and removal of items). Alternate secondary orders are used to extend the existing set of Remote Desktop Protocol (RDP) drawing orders.

1.3.1.2.1 Primary Drawing Orders

Primary orders are mainly used to encode drawing operations. Each primary order is organized into a set of fields to which field-compression algorithms are applied. These algorithms are designed to eliminate sending a field if it has not changed since the last time the order was sent, and to reduce the size of the field encoding for certain field types when they can be represented by smaller sized data.

The following are the broad categories of drawing operations that primary orders encode.

- Combining bitmap patterns using raster operations (called a "blit").
- Drawing graphic objects:
  - Rectangles.
  - Lines.
  - Polygons.
  - Ellipses.

- Displaying text fragments.
- Transferring portions of the screen area from one point to another.
- Temporarily saving obscured regions of the screen.
- Rendering cached bitmaps.
- Rendering cached NineGrid bitmaps.

A subset of the primary drawing operations requires rendering data to be present in a specific cache. For example, to render a NineGrid bitmap, it must be present in the NineGrid cache. This implies that some of the primary drawing orders have a dependency on the secondary orders to first transmit the data necessary to complete a drawing operation to the client.

If a given primary drawing order cannot be constructed, the server must spend time and bandwidth using workarounds, such as using other primary drawing orders or simply sending screen bitmap data in a Bitmap Update (see [MS-RDPBCGR] sections 2.2.9.1.1.3.1.2 and 2.2.9.1.2.1.2). The usage of the primary drawing orders is negotiated by using capability sets.

1.3.1.2.2 Secondary Drawing Orders

Secondary drawing orders are primarily used to manage the addition and removal of items from the following four caches.

- Bitmap Cache
- Color Table Cache
- Glyph Cache
- Brush Cache

Usage of the secondary orders is negotiated through capabilities. If the existence of one of the previous four caches is negotiated, the secondary orders associated with that cache are presumed to be supported.

1.3.1.2.3 Alternate Secondary Drawing Orders

Alternate secondary drawing orders are used to extend the existing set of Remote Desktop Protocol (RDP) drawing orders. The alternate secondary drawing orders described in this document are used to manage offscreen and NineGrid bitmaps and to transport opaque EMF+ records that contain GDI+ 1.1 primitives (as specified in [MS-EMFPLUS] section 2.3.1).

Using these orders, a bitmap can be created in the Offscreen Bitmap Cache, and then set as the target drawing surface on the client. Alternate secondary drawing orders also enable the creation and efficient population of NineGrid bitmaps in the NineGrid bitmap cache. Besides creating and updating entries in these two caches, the alternate secondary orders also support deletion of entries.

GDI+ 1.1 remoting is enabled through the use of alternate secondary drawing orders. In this scenario, the orders are used to transport opaque GDI+ 1.1 records between endpoints, in addition to caching instructions to store GDI+ 1.1 primitives.

Other alternate secondary drawing orders added by RDP extensions are used to support remoting applications (see [MS-RDPERP]) and desktop composition (see [MS-RDPEPC]).

Usage of the alternate secondary orders is negotiated through capabilities.

1.3.1.3 Error Conditions

The Remote Desktop Protocol: GDI Acceleration Extension enables the client to inform the server of errors in the following areas.

- Bitmap caching
- Offscreen bitmap caching
- NineGrid caching
- GDI+ 1.1 remoting

If a client encounters an error in any one of these areas, it can send an appropriate **PDU** to the server to notify it of this issue and ensure appropriate action is taken, such as disabling features within the area.

1.3.2 RDP 6.0 Bulk Compression

The Remote Desktop Protocol: GDI Acceleration Extension builds on the bulk data compression mechanisms described in [MS-RDPBCGR] section 3.1.8 to create a new compressor that employs Huffman encoding algorithms. This new bulk compression technique (known as "RDP 6.0 Bulk Compression") produces higher compression ratios, and, as a result, improves the overall bandwidth consumption of server-to-client traffic (it is not used for client-to-server traffic).

1.3.3 Server Redirection

The Remote Desktop Protocol: GDI Acceleration Extension adds support for redirecting a client connection to a specific session on another server by using the Server Redirection PDU. Using this extension enables basic load-balancing scenarios, as shown in the following figure.

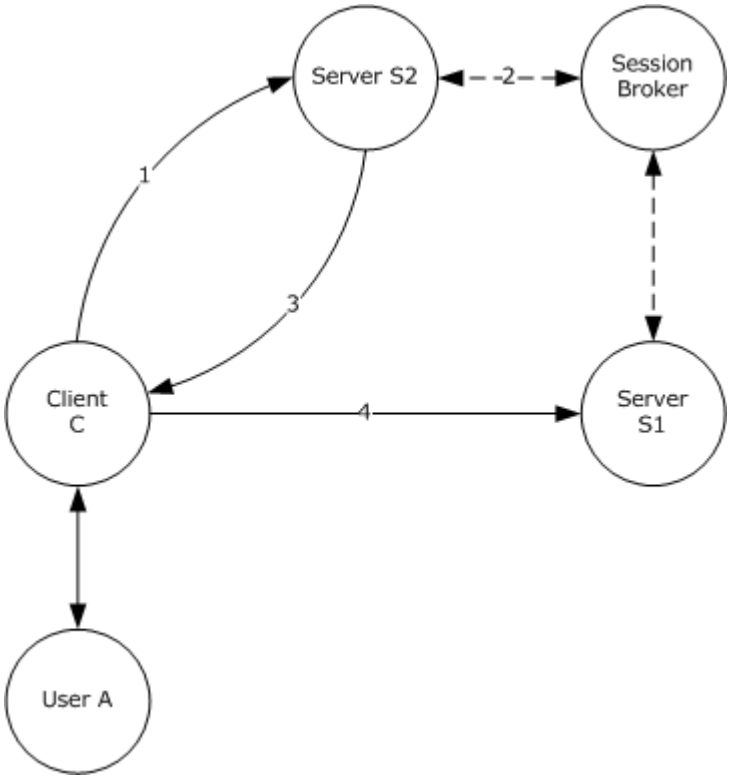


Figure 1: Basic server redirection

Assume that User A has an existing session on Server S1 (Session #3). Both Server S1 and Server S2 are able to communicate with a Session Broker.

1. User A uses Client C to connect to Server S2 and authenticate.
2. Server S2 communicates with the Session Broker and is informed that User A has an existing session on Server S1 (Session #3).
3. Server S2 sends a Redirection PDU to Client C, which contains:
   - The name of the target server (S1)
   - The login credentials to use for Server S1
   - The target Session ID (Session #3)
4. Client C closes the connection to Server S2 and initiates a connection to Server S1. As part of the connection initialization data sent to Server S1, Client C sends the login credentials and requests a connection to Session #3.
5. Server S1 validates the login credentials, and, if they are correct, connects Client C to Session #3.

Besides being used to send the client login credentials, session ID, and target server name, the Server Redirection PDU can also be used to specify the variable-length routing token to place into the X.224 Connection Request PDU of the RDP Connection Sequence (see [MS-RDPBCGR] section 2.2.1.1).

1.4 Relationship to Other Protocols

This protocol extends the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]) by adding advanced drawing order capabilities and compression techniques.

1.5 Prerequisites/Preconditions

All multiple-byte fields within a message are assumed to contain data in little-endian byte ordering, unless otherwise specified.

1.6 Applicability Statement

This protocol is applicable in situations in which it is necessary to optimize the bandwidth required for graphics remoting. The advanced drawing orders specified in this document enable drawing operations to be sent using compact representations, eliminating the need to send bitmaps in many common drawing operations.

1.7 Versioning and Capability Negotiation

The Remote Desktop Protocol: GDI Acceleration Extension builds on the basic Remote Desktop Protocol. All of the features provided by these extensions are negotiated during the Capabilities Negotiation Phase of the RDP Connection Sequence (see [MS-RDPBCGR] section 1.3.1.1). In effect, these extensions merely extend the set of capabilities used by the base Remote Desktop Protocol. (Versioning and capability negotiation in the Remote Desktop Protocol is described in [MS-RDPBCGR] section 1.7.)

1.8 Vendor-Extensible Fields

This protocol contains no vendor-extensible fields.

1.9 Standards Assignments

This protocol makes no standards assignments.

2 Messages

2.1 Transport

The Remote Desktop Protocol: GDI Acceleration Extension packets are an extension to the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting (as specified in [MS-RDPBCGR]) and rely on the same transport. The packets are encapsulated in TCP. The TCP packets MUST be encapsulated in version 4 of the IP protocol.

There is no officially assigned TCP port for the Remote Desktop Protocol: Basic Connectivity and Graphics Remoting, but Remote Desktop Protocol: Basic Connectivity and Graphics Remoting servers listen by default on TCP port 3389 for client requests.

2.2 Message Syntax

2.2.1 Capability Sets

2.2.1.1 Color Table Cache Capability Set (TS_COLORTABLE_CAPABILITYSET)

The TS_COLORTABLE_CAPABILITYSET structure is an unused capability set that advertises the size of the color table cache used in conjunction with the Cache Color Table Secondary Drawing Order (see section 2.2.2.3.1.2.4) and is based on the capability set in [T128] section 8.2.8. This capability is sent by both client and server.

Instead of being specified by the Color Table Capability Set, the existence of color table caching is tied to support for the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing orders. If support for these orders is advertised in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3), the existence of a color table cache with entries for six palettes is implied when palletized color is being used.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
capabilitySetType																lengthCapability															
colorTableCacheSize																pad2octets															

capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_COLORCACHE (0x000A).

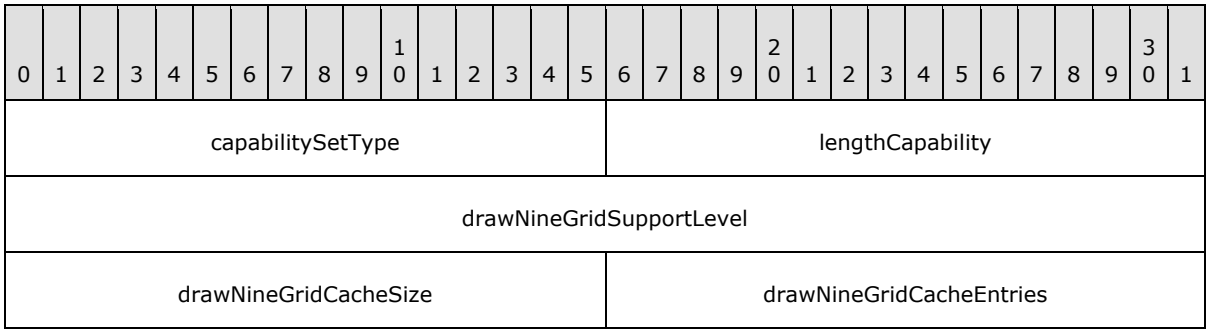
lengthCapability (2 bytes): A 16-bit unsigned integer. The length, in bytes, of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

colorTableCacheSize (2 bytes): A 16-bit unsigned integer. The number of entries in the color table cache (each entry stores a color table). This value MUST be ignored during capability negotiation and is assumed to be 0x0006.

pad2octets (2 bytes): A 16-bit unsigned integer used as padding. Values in this field are arbitrary and MUST be ignored.

2.2.1.2 DrawNineGrid Cache Capability Set (TS_DRAW_NINEGRID_CAPABILITYSET)

The TS_DRAW_NINEGRID_CAPABILITYSET structure is used to advertise support for NineGrid bitmap caching and rendering (see sections 2.2.2.3.1.1.2.21, 2.2.2.3.1.1.2.22, and 2.2.2.3.1.3.4). This capability set is sent only from client to server.



capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_DRAWNINEGRIDCACHE (0x0015).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length, in bytes, of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

drawNineGridSupportLevel (4 bytes): A 32-bit unsigned integer. The level of support for NineGrid drawing. This field MUST be set to one of the following values.

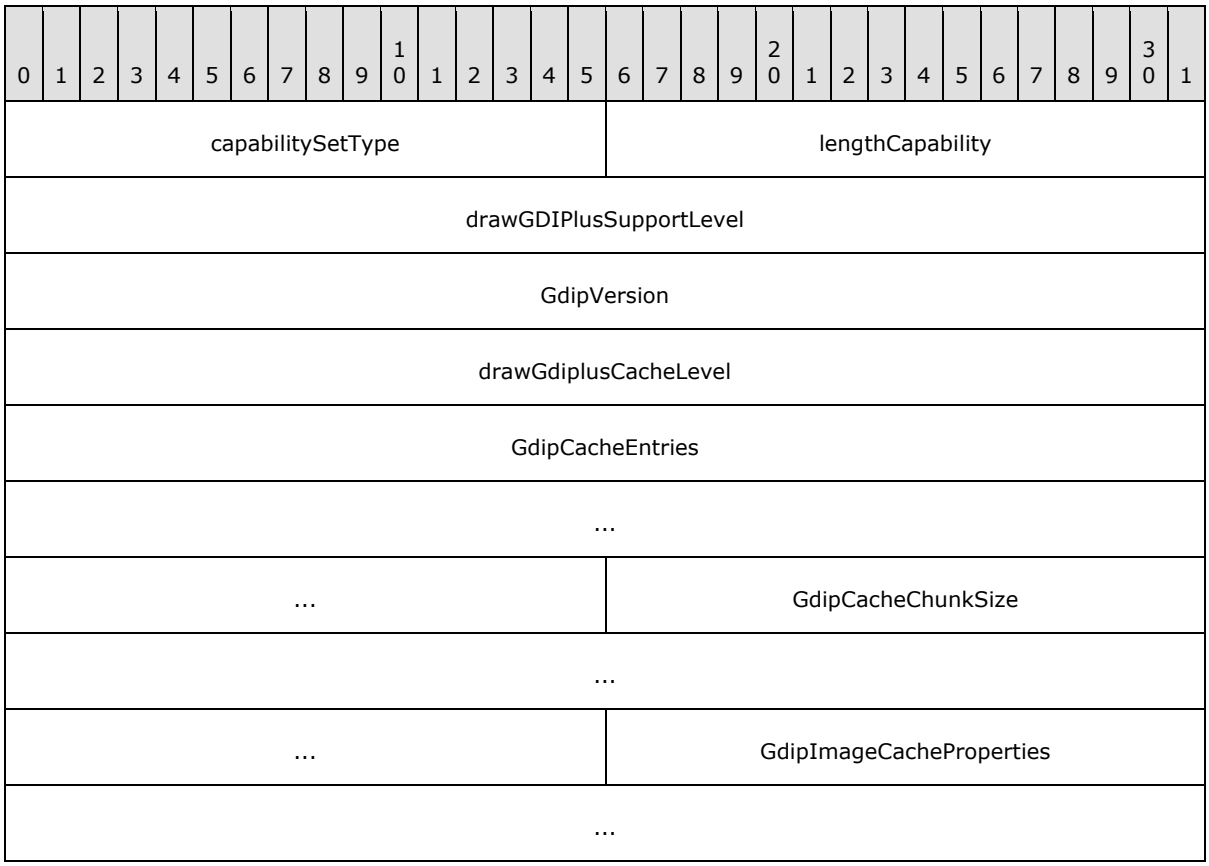
Value	Meaning
DRAW_NINEGRID_NO_SUPPORT 0x00000000	NineGrid bitmap caching and rendering is not supported.
DRAW_NINEGRID_SUPPORTED 0x00000001	Revision 1 NineGrid bitmap caching and rendering is supported. The Revision 1 versions of the stream bitmap alternate secondary orders (see section 2.2.2.3.1.3.5) MUST be used to send the NineGrid bitmap from server to client.
DRAW_NINEGRID_SUPPORTED_REV2 0x00000002	Revision 2 NineGrid bitmap caching and rendering is supported. The Revision 2 versions of the stream bitmap alternate secondary orders (see section 2.2.2.3.1.3.5) MUST be used to send the NineGrid bitmap from server to client.

drawNineGridCacheSize (2 bytes): A 16-bit unsigned integer. The maximum size of the NineGrid Bitmap Cache. The largest size allowed by current RDP servers is 2,560 kilobytes.

drawNineGridCacheEntries (2 bytes): A 16-bit unsigned integer. The maximum number of entries allowed in the NineGrid Bitmap Cache. The maximum number of entries allowed by current RDP servers is 256.

2.2.1.3 Draw GDI+ Capability Set (TS_DRAW_GDIPLUS_CAPABILITYSET)

The TS_DRAW_GDIPLUS_CAPABILITYSET structure is used to advertise the level of GDI+ 1.1 rendering support and the GDI+ cache configuration. This capability is sent by both client and server.



capabilitySetType (2 bytes): A 16-bit unsigned integer. The type of the capability set. This field MUST be set to CAPSTYPE_DRAWGDIPLUS (0x0016).

lengthCapability (2 bytes): A 16-bit unsigned integer. The length, in bytes, of the capability data, including the size of the **capabilitySetType** and **lengthCapability** fields.

drawGDIPlusSupportLevel (4 bytes): A 32-bit unsigned integer. The level of support for GDI+ 1.1 remoting. This field MUST be set to one of the following values.

Value	Meaning
TS_DRAW_GDIPLUS_DEFAULT 0x00000000	GDI+ 1.1 is not supported.
TS_DRAW_GDIPLUS_SUPPORTED 0x00000001	GDI+ 1.1 is supported.

GdipVersion (4 bytes): A 32-bit unsigned integer. The build number of the underlying GDI+ 1.1 subsystem.

drawGdiplusCacheLevel (4 bytes): A 32-bit unsigned integer. The level of support for the caching of GDI+ 1.1 rendering primitives. This field MUST be set to one of the following values.

Value	Meaning
TS_DRAW_GDIPLUS_CACHE_LEVEL_DEFAULT 0x00000000	Caching of GDI+ 1.1 rendering primitives is not supported.
TS_DRAW_GDIPLUS_CACHE_LEVEL_ONE 0x00000001	Caching of GDI+ 1.1 rendering primitives is supported.

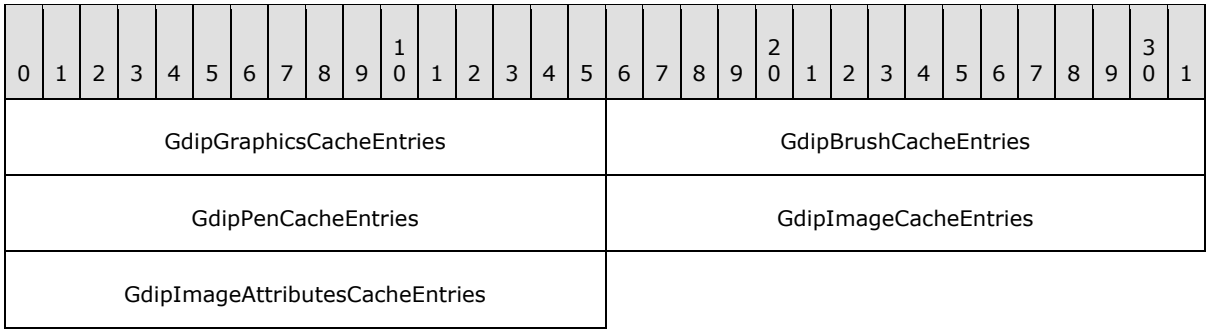
GdipCacheEntries (10 bytes): A GDI+ Cache Entries (section 2.2.1.3.1) structure that specifies the total number of entries within the GDI+ Graphics, Pen, Brush, Image, and Image Attributes caches.

GdipCacheChunkSize (8 bytes): A GDI+ Cache Chunk Size structure that specifies the size of individual entries in the GDI+ Graphics, Brush, Pen, and Image Attributes caches.

GdipImageCacheProperties (6 bytes): A GDI+ Image Cache Properties structure that contains sizing information for the GDI+ Image cache.

2.2.1.3.1 GDI+ Cache Entries (TS_GDIPLUS_CACHE_ENTRIES)

The TS_GDIPLUS_CACHE_ENTRIES structure specifies the total number of cache entries for the GDI+ Graphics, Brush, Pen, Image, and Image Attributes caches.



GdipGraphicsCacheEntries (2 bytes): A 16-bit unsigned integer. The total number of entries allowed in the GDI+ Graphics cache. The maximum allowed value is 10 entries.

GdipBrushCacheEntries (2 bytes): A 16-bit unsigned integer. The total number of entries allowed in the GDI+ Brush cache. The maximum allowed value is 5 entries.

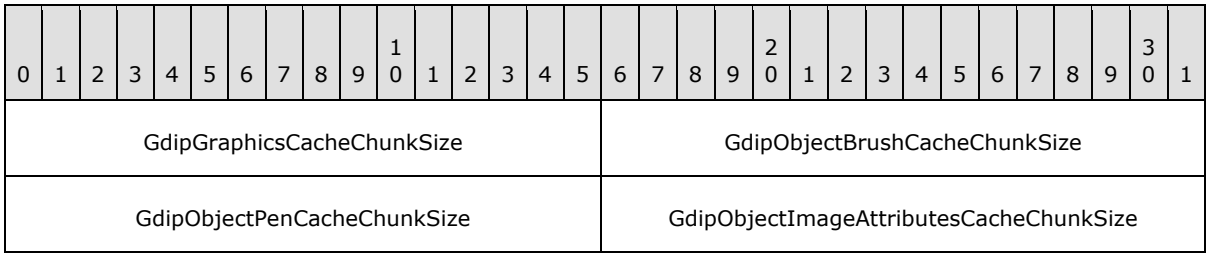
GdipPenCacheEntries (2 bytes): A 16-bit unsigned integer. The total number of entries allowed in the GDI+ Pen cache. The maximum allowed value is 5 entries.

GdipImageCacheEntries (2 bytes): A 16-bit unsigned integer. The total number of entries allowed in the GDI+ Image cache. The maximum allowed value is 10 entries.

GdipImageAttributesCacheEntries (2 bytes): A 16-bit unsigned integer. The total number of entries allowed in the GDI+ Image Attributes cache. The maximum allowed value is 2 entries.

2.2.1.3.2 GDI+ Cache Chunk Size (TS_GDIPLUS_CACHE_CHUNK_SIZE)

The TS_GDIPLUS_CACHE_CHUNK_SIZE structure specifies the size of individual entries in the GDI+ Graphics, Brush, Pen, and Image Attributes caches.



GdipGraphicsCacheChunkSize (2 bytes): A 16-bit unsigned integer. The size in bytes of a GDI+ Graphics cache entry. The maximum allowed value is 512 bytes.

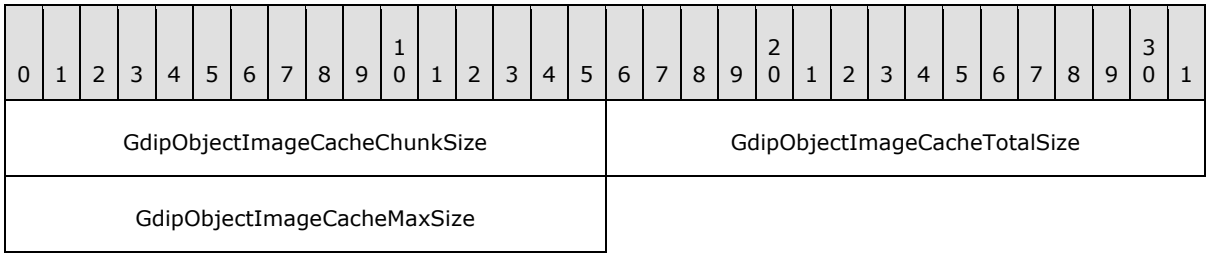
GdipObjectBrushCacheChunkSize (2 bytes): A 16-bit unsigned integer. The size in bytes of a GDI+ Brush cache entry. The maximum allowed value is 2,048 bytes.

GdipObjectPenCacheChunkSize (2 bytes): A 16-bit unsigned integer. The size in bytes of a GDI+ Pen cache entry. The maximum allowed value is 1,024 bytes.

GdipObjectImageAttributesCacheChunkSize (2 bytes): A 16-bit unsigned integer. The size in bytes of a GDI+ Image Attributes cache entry. The maximum allowed value is 64 bytes.

2.2.1.3.3 GDI+ Image Cache Properties (TS_GDIPLUS_IMAGE_CACHE_PROPERTIES)

The TS_GDIPLUS_IMAGE_CACHE_PROPERTIES structure contains sizing information for the GDI+ Image cache.



GdipObjectImageCacheChunkSize (2 bytes): A 16-bit unsigned integer. The size, in bytes, of a chunk in the GDI+ Image cache. The maximum allowed value is 4,096 bytes.

GdipObjectImageCacheTotalSize (2 bytes): A 16-bit unsigned integer. The total number of chunks in the GDI+ Image cache. The maximum allowed value is 256 chunks.

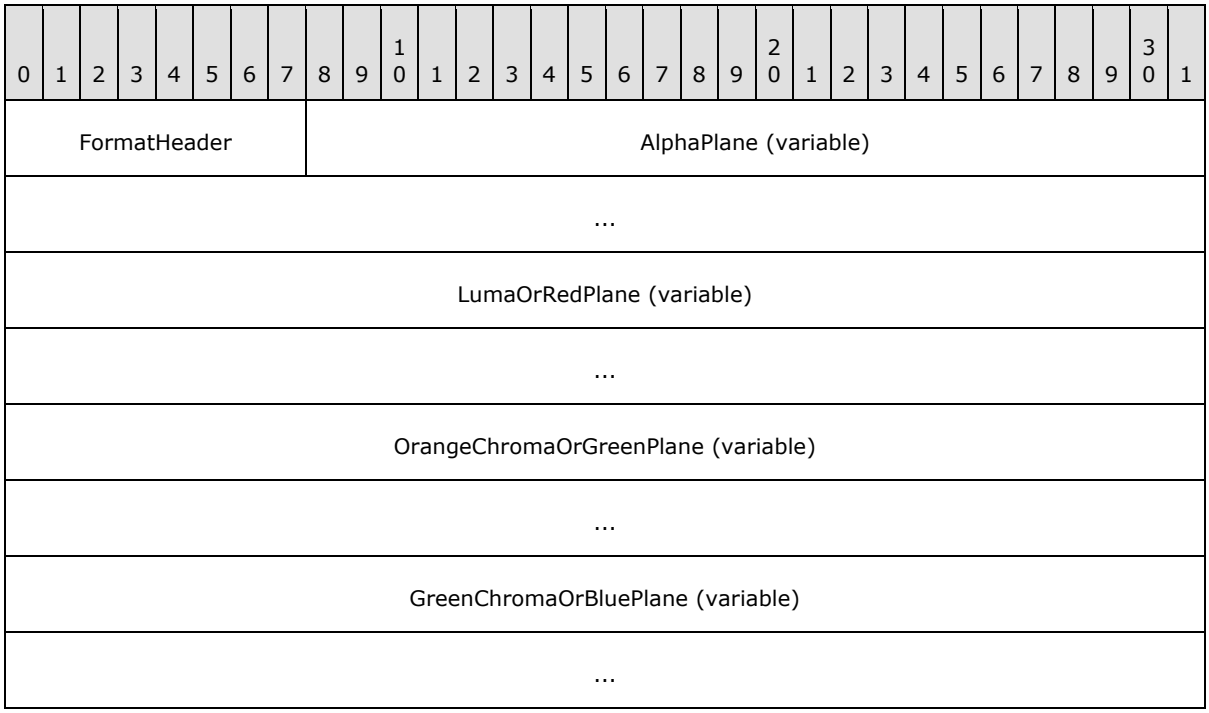
GdipObjectImageCacheMaxSize (2 bytes): A 16-bit unsigned integer. The total number of chunks that can be used by an entry in the GDI+ Image cache. The default value is 128 chunks.

2.2.2 Accelerated Graphics

2.2.2.1 RDP 6.0 Bitmap Compressed Bitmap Stream (RDP6_BITMAP_STREAM)

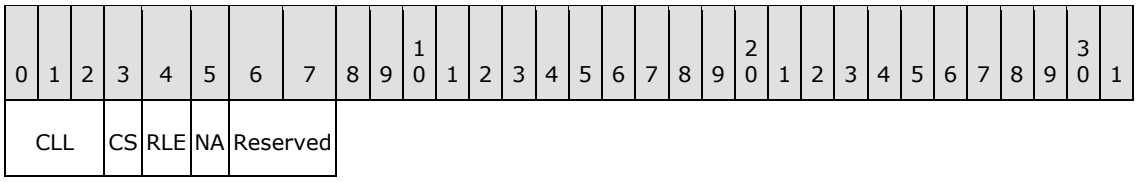
The RDP6_BITMAP_STREAM structure contains a stream of bitmap data compressed using the RDP 6.0 Bitmap Compression techniques (see section 3.1.9). Depending on the compression techniques employed, the bitmap data is represented using the AYCoCg or ARGB color space (see section 3.1.9.1.2).

Compressed bitmap data is sent as part of the Bitmap Update (see [MS-RDPBCGR] section 2.2.9.1.1.3.1.2), Fast-Path Bitmap Update (see [MS-RDPBCGR] section 2.2.9.1.2.1.2), Cache Bitmap - Revision 1 (section 2.2.2.3.1.2.2) Secondary Drawing Orders, or Cache Bitmap - Revision 2 (section 2.2.2.3.1.2.3) Secondary Drawing Orders. In all of these cases, the data is encapsulated inside a Bitmap Data structure (as specified in [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.3).



FormatHeader (1 byte): An 8-bit unsigned integer. This field contains a one byte bit-packed update header as follows.

The format of the updateHeader byte is described by the following bitmask diagram.



CLL (3 bits): A 3-bit field. Indicates the Color Loss Level (see section 3.1.9.1.4). If **CLL** is set to 0, the color space used is ARGB. Otherwise, **CLL** MUST be in the range 1 to 7 (inclusive), and the color space used is AYCoCg.

CS (1 bit): A 1-bit field. If **CS** is equal to 1, Chroma Subsampling (section 3.1.9.1.3) is used, and the **CLL** field MUST be greater than 0, as Chroma Subsampling applies only to the AYCoCg color space.

RLE (1 bit): A 1-bit field. If **RLE** is equal to 1, RDP 6.0 RLE is used to compress the color planes (see section 3.1.9.2). If not, **RLE** is equal to 0, and the color plane is sent uncompressed.

NA (1 bit): A 1-bit field. Indicates if an alpha plane is present. If **NA** is equal to 1, there is no alpha plane. The values of the alpha plane are then assumed to be 0xFF (fully opaque), and the bitmap data contains only three color planes. If **NA** is equal to 0, the alpha plane is sent as the first color plane.

Reserved (2 bits): A 2-bit field. Reserved for future use.

AlphaPlane (variable): A variable-length field. If the **RLE** subfield in the FormatHeader indicates that all of the color planes are RLE compressed (see section 3.1.9.2), this field contains an RLE Segments (section 2.2.2.1.1) structure. Otherwise, it contains the raw bytes of the color plane.

LumaOrRedPlane (variable): A variable-length field. Contains the luma plane (AYCoCg color space) or red plane (ARGB color space). If the **CLL** subfield of the FormatHeader is greater than 0, the AYCoCg color space MUST be used. Otherwise, the ARGB color space MUST be used.

If the **RLE** subfield in the FormatHeader indicates that all of the color planes are RLE compressed (see section 3.1.9.2), this field contains an RLE Segments (section 2.2.2.1.1) structure. Otherwise, it contains the raw bytes of the color plane.

Depending on the values of the **CLL** and **CS** subfields of the FormatHeader (in the case of the AYCoCg color space), the luma plane may have been transformed by Color Loss Reduction (section 3.1.9.1.4) and Chroma Subsampling (section 3.1.9.1.3).

OrangeChromaOrGreenPlane (variable): A variable-length field. Orange chroma plane (AYCoCg color space) or green plane (ARGB color space). If the **CLL** subfield of the FormatHeader is greater than 0, the AYCoCg color space MUST be used. Otherwise, the ARGB color space MUST be used.

If the **RLE** subfield in the FormatHeader indicates that all of the color planes are RLE compressed (see section 3.1.9.2), this field contains an RLE Segments (section 2.2.2.1.1) structure. Otherwise, it contains the raw bytes of the color plane.

Depending on the values of the **CLL** and **CS** subfields of the FormatHeader (in the case of the AYCoCg color space), the orange chroma plane may have been transformed by Color Loss Reduction (section 3.1.9.1.4) and Chroma Subsampling (section 3.1.9.1.3).

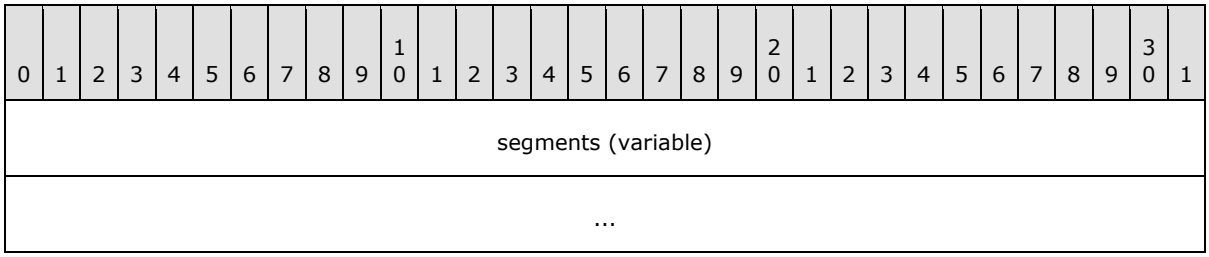
GreenChromaOrBluePlane (variable): A variable-length field. Green chroma plane (AYCoCg color space) or blue plane (ARGB color space). If the **CLL** subfield of the FormatHeader is greater than 0, the AYCoCg color space MUST be used. Otherwise, the ARGB color space MUST be used.

If the **RLE** subfield in the FormatHeader indicates that all of the color planes are RLE compressed (see section 3.1.9.2), this field contains an RLE Segments (section 2.2.2.1.1) structure. Otherwise, it contains the raw bytes of the color plane.

Depending on the values of the **CLL** and **CS** subfields of the FormatHeader (in the case of the AYCoCg color space), the green chroma plane may have been transformed by Color Loss Reduction (section 3.1.9.1.4) and Chroma Subsampling (section 3.1.9.1.3).

2.2.2.1.1 RDP 6.0 RLE Segments (RDP6_RLE_SEGMENTS)

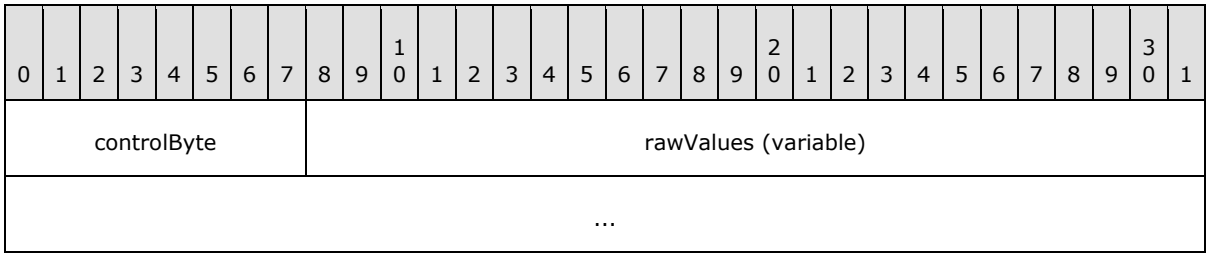
The RDP6_RLE_SEGMENTS structure contains the run-length encoded contents of a color plane and consists of a collection of RDP6_RLE_SEGMENT structures.



segments (variable): This field contains a variable-length array of RDP 6.0 RLE Segment (section 2.2.2.1.2) structures.

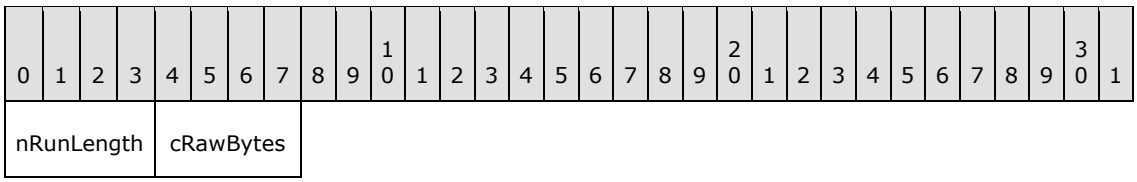
2.2.2.1.2 RDP 6.0 RLE Segment (RDP6_RLE_SEGMENT)

The RDP6_RLE_SEGMENT structure encodes an RLE segment that contains a RAW and RUN component (see section 3.1.9.2).



controlByte (1 byte): An 8-bit unsigned integer. Contains the RAW and RUN components of an RDP 6.0 RLE segment (see section 3.1.9.2). The **controlByte** field MUST contain a nonzero value.

The format of the **controlByte**byte is described by the following bitmask diagram.



nRunLength (4 bits): A 4-bit field. The number of times the last RAW byte in the **rawValues** field is repeated (known as the run-length).

Because a RUN MUST be a sequence of at least three values (see section 3.1.9.2), the values 1 and 2 are used in the **nRunLength** field to encode extra long RUN sequences of more than 16 values.

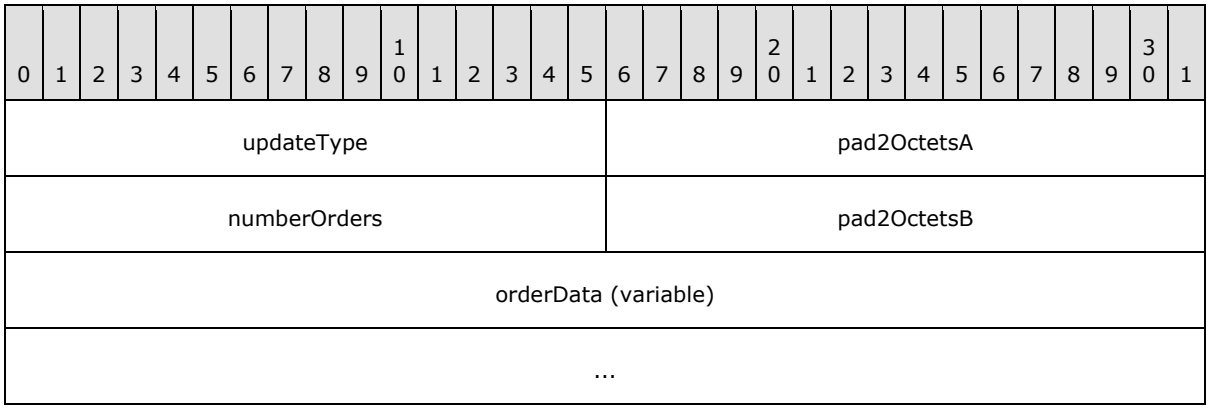
- If the **nRunLength** field is set to 1, the actual run-length is 16 plus the value in **cRawBytes**. On decode, the number of RAW bytes in the **rawValues** field is assumed to be zero. This gives a maximum run-length of 31 values.
- If the **nRunLength** field is set to 2, the actual run-length is 32 plus the value in **cRawBytes**. On decode, the number of RAW bytes in the **rawValues** field is assumed to be zero. This gives a maximum run-length of 47 values.

cRawBytes (4 bits): A 4-bit field. The number of RAW bytes in the **rawValues** field.

rawValues (variable): A variable-length field that contains the actual RAW bytes representing a portion of the color plane being encoded. If the bytes belong to the first scan line, they represent absolute values. Otherwise, the bytes represent delta values (see section 3.1.9.2).

2.2.2.2 Orders Update (TS_UPDATE_ORDERS_PDU_DATA)

The TS_UPDATE_ORDERS_PDU_DATA structure contains primary, secondary, and alternate secondary drawing orders aligned on byte boundaries. This structure conforms to the layout of a Slow Path Graphics Update (see [MS-RDPBCGR] section 2.2.9.1.1.3.1) and is encapsulated within a Graphics Update PDU (see [MS-RDPBCGR] section 2.2.9.1.1.3).



updateType (2 bytes): A 16-bit unsigned integer. The field contains the graphics update type. This field MUST be set to UPDATETYPE_ORDERS (0x0000).

pad2OctetsA (2 bytes): A 16-bit unsigned integer used as a padding field. Values in this field are arbitrary and MUST be ignored.

numberOrders (2 bytes): A 16-bit unsigned integer. The number of Drawing Order (section 2.2.2.2.1) structures contained in the **orderData** field.

pad2OctetsB (2 bytes): A 16-bit unsigned integer used as a padding field. Values in this field are arbitrary and MUST be ignored.

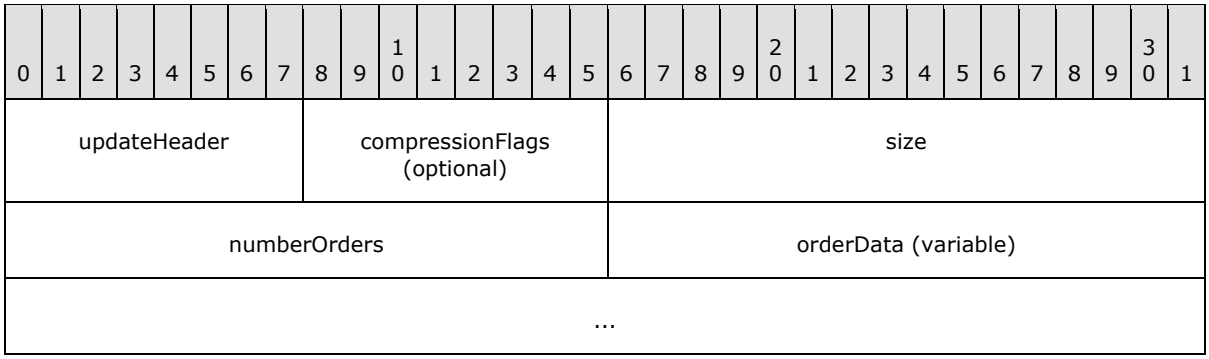
orderData (variable): A variable sized array of Drawing Order (section 2.2.2.2.1) structures packed on byte boundaries. Each structure contains a primary, secondary, or alternate secondary drawing order. The **controlFlags** field of the Drawing Order identifies the type of drawing order.

2.2.2.2.1 Drawing Order (DRAWING_ORDER)

The DRAWING_ORDER structure encapsulated by the Orders Update (section 2.2.2.2) is identical to the DRAWING_ORDER structure used in conjunction with the Fast-Path Orders Update (section 2.2.2.3).

2.2.2.3 Fast-Path Orders Update (TS_FP_UPDATE_ORDERS)

The TS_FP_UPDATE_ORDERS structure contains primary, secondary, and alternate secondary drawing orders aligned on byte boundaries. This structure conforms to the layout of a Fast-Path Update (see [MS-RDPBCGR] section 2.2.9.1.2.1) and is encapsulated within a Fast-Path Update PDU (see [MS-RDPBCGR] section 2.2.9.1.2).



updateHeader (1 byte): An 8-bit unsigned integer. The format of this field is the same as the **updateHeader** byte field described in the Fast-Path Update structure (see [MS-RDPBCGR] section 2.2.9.1.2.1). The **updateCode** bitfield (4 bits in size) MUST be set to FASTPATH_UPDATETYPE_ORDERS (0x0).

compressionFlags (1 byte): An 8-bit unsigned integer. The format of this optional field (as well as the possible values) is the same as the **compressionFlags** field described in the Fast-Path Update structure specified in [MS-RDPBCGR] section 2.2.9.1.2.1.

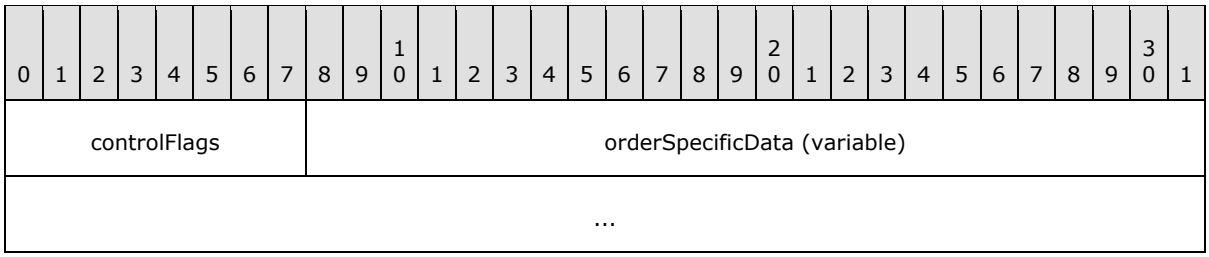
size (2 bytes): A 16-bit unsigned integer. The format of this field (as well as the possible values) is the same as the **size** field described in the Fast-Path Update structure specified in [MS-RDPBCGR] section 2.2.9.1.2.1.

numberOrders (2 bytes): A 16-bit unsigned integer. The number of Drawing Order (section 2.2.2.2.1) structures contained in the **orderData** field.

orderData (variable): A variable sized array of Drawing Order (section 2.2.2.2.1) structures packed on byte boundaries. Each structure contains a primary, secondary, or alternate secondary drawing order. The **controlFlags** field of the Drawing Order identifies the type of drawing order.

2.2.2.3.1 Drawing Order (DRAWING_ORDER)

The DRAWING_ORDER structure is used to describe and encapsulate a single primary, secondary, or alternate secondary drawing order sent from server to client. All drawing orders conform to this basic structure (see sections 2.2.2.3.1.1.2, 2.2.2.3.1.2.1.1, and 2.2.2.3.1.3.1.1).



controlFlags (1 byte): An 8-bit unsigned integer. A control byte that identifies the class of the drawing order.

If the TS_STANDARD (0x01) flag is set, the order is a primary drawing order. If both the TS_STANDARD (0x01) and TS_SECONDARY (0x02) flags are set, the order is a secondary drawing order. Finally, if only the TS_SECONDARY (0x02) flag is set, the order is an alternate secondary drawing order.

More flags MAY be present, depending on the drawing order class. The flags listed are common to all three classes of drawing orders.

Name	Value
TS_STANDARD	0x01
TS_SECONDARY	0x02

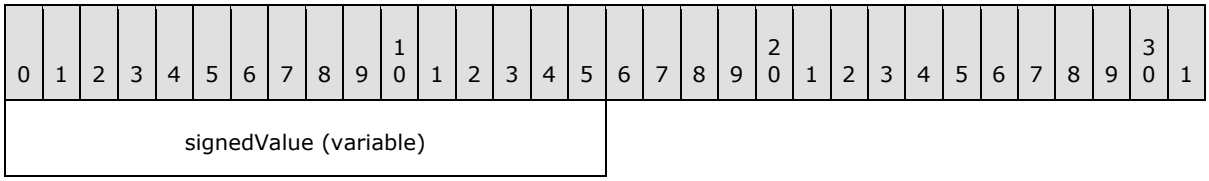
orderSpecificData (variable): Variable-length data specific to the drawing order class and the drawing order itself.

2.2.2.3.1.1 Primary Drawing Orders

2.2.2.3.1.1.1 Common Data Types

2.2.2.3.1.1.1.1 Coord Field (COORD_FIELD)

The COORD_FIELD structure is used to describe a value in the range -32768 to 32767.



signedValue (2 bytes): A signed one-byte or two-byte value that describes a coordinate in the range -32768 to 32767.

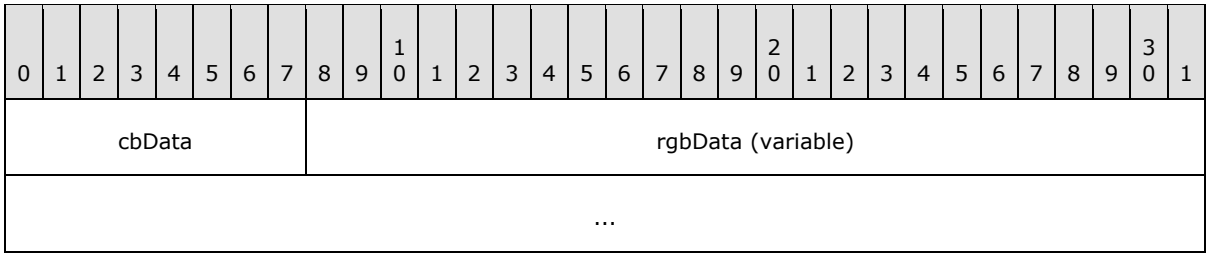
When the **controlFlags** field (see section 2.2.2.3.1.1.2) of the primary drawing order that contains the COORD_FIELD structure has the DELTA_COORDINATES flag (0x10) set, the **signedValue** field MUST contain a signed one-byte value. If the DELTA_COORDINATES flag is not set, the **signedValue** field MUST contain a two-byte signed value.

The one-byte format contains a signed delta from the previous value of the Coord field. To obtain the new value of the field, the decoder MUST increment the previous value of the field

by the signed delta to produce the current value. The two-byte format is simply the full value of the field that MUST replace the previous value.

2.2.2.3.1.1.1.2 One-Byte Header Variable Field (VARIABLE1_FIELD)

TheVARIABLE1_FIELD structure is used to encode a variable-length byte-stream that will hold a maximum of 255 bytes. This structure is always situated at the end of an order.

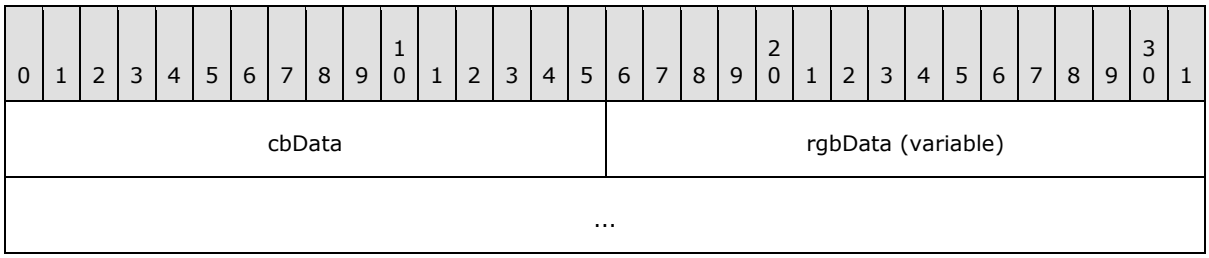


cbData (1 byte): An 8-bit unsigned integer. The number of bytes present in the **rgbData** field.

rgbData (variable): Variable-length binary data. The size of this data, in bytes, is given by the **cbData** field.

2.2.2.3.1.1.1.3 Two-Byte Header Variable Field (VARIABLE2_FIELD)

The VARIABLE2_FIELD structure is used to encode a variable-length byte-stream that holds a maximum of 32,767 bytes. This structure is always situated at the end of an order.

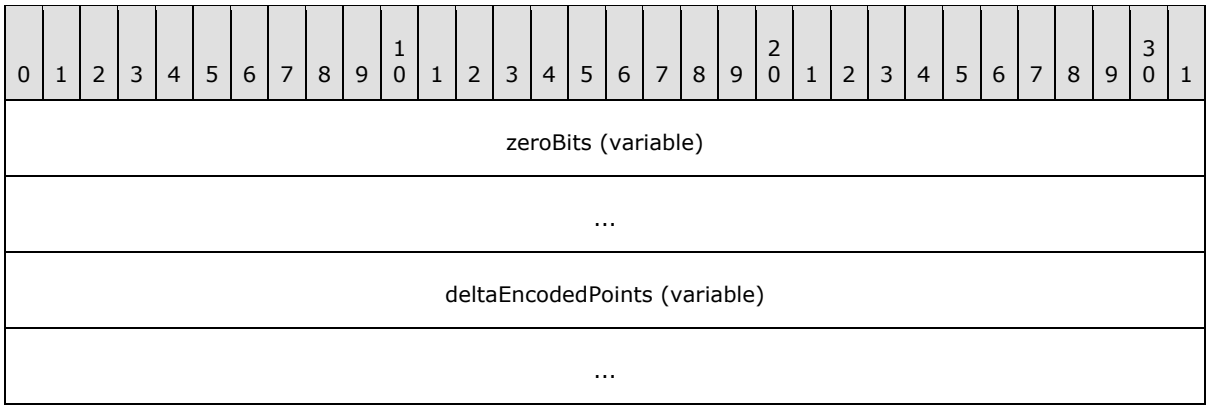


cbData (2 bytes): A 16-bit unsigned integer. The number of bytes present in the **rgbData** field.

rgbData (variable): Variable-length binary data. The size of this data, in bytes, is given by the **cbData** field.

2.2.2.3.1.1.1.4 Delta-Encoded Points (DELTA_PTS_FIELD)

The DELTA_PTS_FIELD structure is used to encode a series of points. Each point is expressed as an X and Y delta from the previous point in the series (the first X and Y deltas are relative to a base point that MUST be included in the order that contains the DELTA_PTS_FIELD structure). The number of points is order-dependent, and is not specified by any field within the DELTA_PTS_FIELD structure. Instead, a separate field within the order that contains the DELTA_PTS_FIELD structure MUST be used to specify the number of points (this field SHOULD<1> be placed immediately before the DELTA_PTS_FIELD structure in the order encoding).



zeroBits (variable): A variable-length byte field. The **zeroBits** field is used to indicate the absence of an X or Y delta value for a specific point in the series. The size in bytes of the **zeroBits** field is given by ceil(NumPoints / 4) where NumPoints is the number of points being encoded. Each point in the series requires two zero-bits (four points per byte) to indicate whether an X or Y delta value is zero (and not present), starting with the most significant bits, so that for the first point the X-zero flag is set at (zeroBits[0] & 0x80), and the Y-zero flag is set at (zeroBits[0] & 0x40).

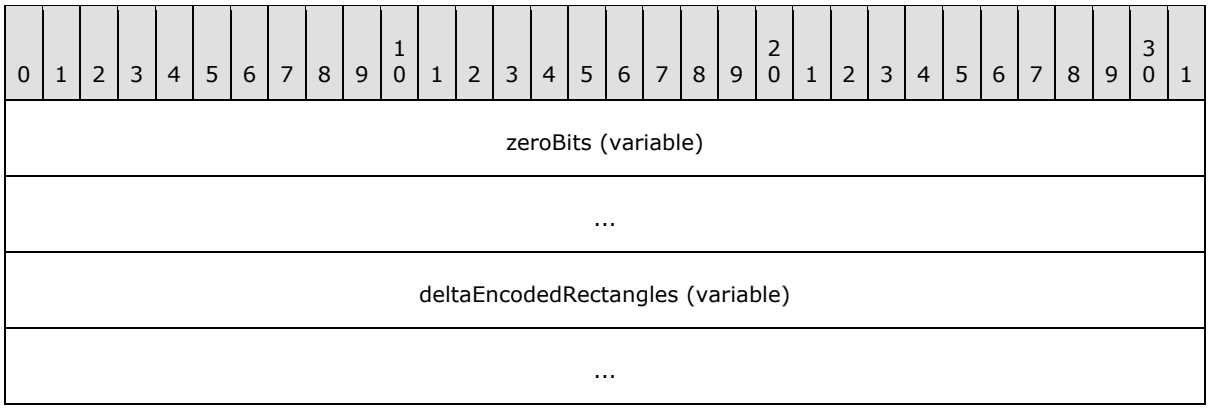
deltaEncodedPoints (variable): A variable-length byte field. The **deltaEncodedPoints** field contains a series of (X, Y) pairs, each pair specifying the delta from the previous pair in the series (the first pair in the series contains a delta from a pre-established coordinate).

The presence of the X and Y delta values for a given pair in the series is dictated by the individual bits of the **zeroBits** field. If the zero bit is set for a given X or Y component, its value is unchanged from the previous X or Y component in the series (a delta of zero), and no data is provided. If the zero bit is not set for a given X or Y component, the delta value it represents is encoded in a packed signed format:

- If the high bit (0x80) is not set in the first encoding byte, the field is one byte long and is encoded as a signed delta in the lower seven bits of the byte.
- If the high bit of the first encoding byte is set, the lower 7 bits of the first byte and the 8 bits of the next byte are concatenated (the first byte containing the high order bits) to create a 15-bit signed delta value.

2.2.2.3.1.1.1.5 Delta-Encoded Rectangles (DELTA_RECTS_FIELD)

The DELTA_RECTS_FIELD structure is used to encode a series of rectangles. Each rectangle is encoded as a (left, top, width, height) quadruple with the left and top components of each quadruple containing the delta from the left and top components of the previous rectangle; the first rectangle in the series is implicitly assumed to be (0, 0, 0, 0). The number of rectangles is order-dependent, and is not specified by any field within the DELTA_RECTS_FIELD structure. Instead, a separate field within the order that contains the DELTA_RECTS_FIELD structure MUST be used to specify the number of rectangles (this field SHOULD be placed immediately before the DELTA_RECTS_FIELD structure in the order encoding). The maximum number of rectangles that can be encoded by this structure is 45.



zeroBits (variable): A variable-length byte field. The **zeroBits** field is used to indicate the absence of a left, top, width, or height component. The size, in bytes, of the **zeroBits** field is given by ceil(NumRects / 2) where NumRects is the number of rectangles being encoded. Each rectangle in the series requires four zero-bits (two rectangles per byte) to indicate whether a left, top, width, or height component is zero (and not present), starting with the most significant bits, so that for the first rectangle the left-zero flag is set at (**zeroBits**[0] & 0x80), the top-zero flag is set at (**zeroBits**[0] & 0x40), the width-zero flag is set at (**zeroBits**[0] & 0x20), and the height-zero flag is set at (**zeroBits**[0] & 0x10).

deltaEncodedRectangles (variable): A variable-length byte field. The **deltaEncodedRectangles** field contains a series of (left, top, width, height) quadruples with the left and top components in each quadruple specifying the delta from the left and top components of the previous rectangle in the series; the first rectangle in the series is implicitly assumed to be (0, 0, 0, 0).

Assume there are two rectangles specified using (left, top, right, bottom) quadruples:

```
1: (L1, T1, R1, B1)
2: (L2, T2, R2, B2)
```

Assuming Rectangle 1 is the first in the series, and using the (left, top, width, height) quadruple encoding scheme, these two rectangles would be specified as:

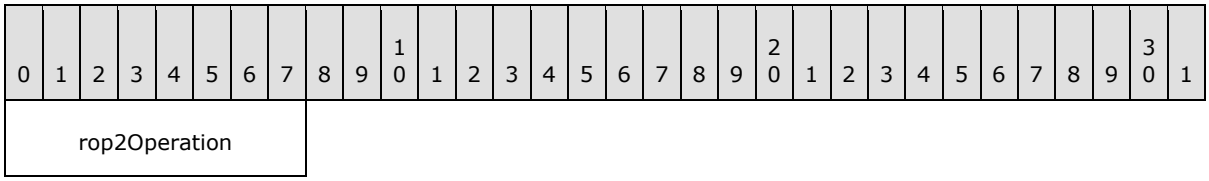
```
1: (L1, T1, R1 - L1, B1 - T1)
2: (L2 - L1, T2 - T1, R2 - L2, B2 - T2)
```

The presence of the left, top, width, or height component for a given quadruple is dictated by the individual bits of the **zeroBits** field. If the zero bit is set for a given left, top, width, or height component, its value is unchanged from the previous corresponding left, top, width, or height value in the series (a delta of zero), and no data is provided. If the zero bit is not set for a left, right, width, or height component, its value is encoded in a packed signed format:

- If the high bit (0x80) is not set in the first encoding byte, the field is one byte long and is encoded as a signed delta in the lower seven bits of the byte.
- If the high bit of the first encoding byte is set, the lower 7 bits of the first byte and the 8 bits of the next byte are concatenated (the first byte containing the high order bits) to create a 15-bit signed delta value.

2.2.2.3.1.1.1.6 Binary Raster Operation (ROP2_OPERATION)

The ROP2_OPERATION structure is used to define how the bits in a destination bitmap and a selected brush pen must be combined using Boolean operators.



rop2Operation (1 byte): An 8-bit unsigned integer. A raster-operation code that describes a Boolean operation, in **Reverse Polish Notation**, to perform on the bits in a destination bitmap (D) and selected brush or pen (P). This operation is a combination of the AND (a), OR (o), NOT (n), and XOR (x) Boolean operators.

Value	Meaning
R2_BLACK 0x01	0
R2_NOTMERGEPEN 0x02	DPon
R2_MASKNOTPEN 0x03	DPna
R2_NOTCOPYPEN 0x04	Pn
R2_MASKPENNOT 0x05	PDna
R2_NOT 0x06	Dn
R2_XORPEN 0x07	DPx
R2_NOTMASKPEN 0x08	DPan
R2_MASKPEN 0x09	DPa
R2_NOTXORPEN 0x0A	DPon
R2_NOP 0x0B	D
R2_MERGENOTPEN 0x0C	DPno
R2_COPYPEN 0x0D	P
R2_MERGEPENNOT	PDno

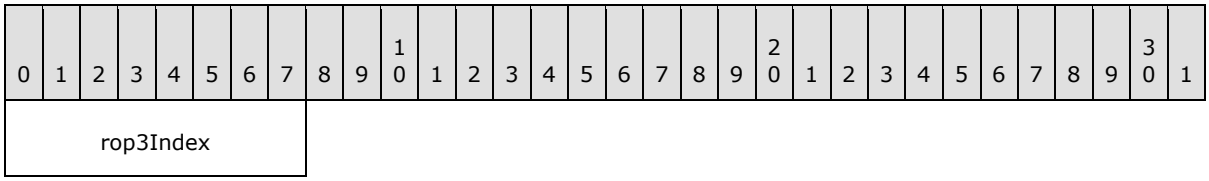
Value	Meaning
0x0E	
R2_MERGEPEN 0x0F	PDo
R2_WHITE 0x10	1

For example, by using the previous table, one can derive that the R2_MERGEPEN (0x0F) operation replaces the values of the pixels in the destination bitmap with a combination of pixel values of the destination and pen.

For more information about binary raster operations, see [MSDN-BRO] and [WGFX] section 8.1.

2.2.2.3.1.1.1.7 Ternary Raster Operation Index (ROP3_OPERATION_INDEX)

The ROP3_OPERATION_INDEX structure is used to define how the bits in a source bitmap, destination bitmap, and a selected brush or pen must be combined using Boolean operators.



rop3Index (1 byte): An 8-bit unsigned integer. This field contains an index of a raster operation code that describes a Boolean operation, in Reverse Polish Notation, to perform on the bits in a source bitmap (S), destination bitmap (D), and selected brush or pen (P). This operation is a combination of the AND (a), OR (o), NOT (n), and XOR (x) Boolean operators.

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x00	BLACKNESS 0x00000042	0
0x01	0x00010289	DPSoon
0x02	0x00020C89	DPSona
0x03	0x000300AA	PSon
0x04	0x00040C88	SDPona
0x05	0x000500A9	DPon
0x06	0x00060865	PDSxnon
0x07	0x000702C5	PDSaon
0x08	0x00080F08	SDPnaa
0x09	0x00090245	PDSxon

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x0A	0x000A0329	DPna
0x0B	0x000B0B2A	PSDnaon
0x0C	0x000C0324	SPna
0x0D	0x000D0B25	PDSnaon
0x0E	0x000E08A5	PDSonon
0x0F	0x000F0001	Pn
0x10	0x00100C85	PDSona
0x11	NOTSRCERASE 0x001100A6	DSon
0x12	0x00120868	SDPxnon
0x13	0x001302C8	SDPaon
0x14	0x00140869	DPSxnon
0x15	0x001502C9	DPSaon
0x16	0x00165CCA	PSDPSanaxx
0x17	0x00171D54	SSPxDSxaxn
0x18	0x00180D59	SPxPDxa
0x19	0x00191CC8	SDPSanaxn
0x1A	0x001A06C5	PDSPaox
0x1B	0x001B0768	SDPSxaxn
0x1C	0x001C06CA	PSDPaox
0x1D	0x001D0766	DSPDxaxn
0x1E	0x001E01A5	PDSox
0x1F	0x001F0385	PDSoan
0x20	0x00200F09	DPSnaa
0x21	0x00210248	SDPxon
0x22	0x00220326	DSna
0x23	0x00230B24	SPDnaon
0x24	0x00240D55	SPxDSxa
0x25	0x00251CC5	PDSPanaxn
0x26	0x002606C8	SDPSaox

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x27	0x00271868	SDPSxnox
0x28	0x00280369	DPSxa
0x29	0x002916CA	PSDPSaoxxn
0x2A	0x002A0CC9	DPSana
0x2B	0x002B1D58	SSPxPDxaxn
0x2C	0x002C0784	SPDSoax
0x2D	0x002D060A	PSDnox
0x2E	0x002E064A	PSDPxox
0x2F	0x002F0E2A	PSDnoan
0x30	0x0030032A	PSna
0x31	0x00310B28	SDPnaon
0x32	0x00320688	SDPSoox
0x33	NOTSRCCOPY 0x00330008	Sn
0x34	0x003406C4	SPDSaox
0x35	0x00351864	SPDSxnox
0x36	0x003601A8	SDPox
0x37	0x00370388	SDPoan
0x38	0x0038078A	PSDPoax
0x39	0x00390604	SPDnox
0x3A	0x003A0644	SPDSxox
0x3B	0x003B0E24	SPDnoan
0x3C	0x003C004A	PSx
0x3D	0x003D18A4	SPDSonox
0x3E	0x003E1B24	SPDSnaox
0x3F	0x003F00EA	PSan
0x40	0x00400F0A	PSDnaa
0x41	0x00410249	DPSxon
0x42	0x00420D5D	SDxPDxa
0x43	0x00431CC4	SPDSanaxn

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x44	SRCERASE 0x00440328	SDna
0x45	0x00450B29	DPSnaon
0x46	0x004606C6	DSPDaox
0x47	0x0047076A	PSDPxaxn
0x48	0x00480368	SDPxa
0x49	0x004916C5	PDSPDaoxxn
0x4A	0x004A0789	DPSDoax
0x4B	0x004B0605	PDSnox
0x4C	0x004C0CC8	SDPana
0x4D	0x004D1954	SSPxDSxoxn
0x4E	0x004E0645	PDSPxox
0x4F	0x004F0E25	PDSnoan
0x50	0x00500325	PDna
0x51	0x00510B26	DSPnaon
0x52	0x005206C9	DPSDaox
0x53	0x00530764	SPDSxaxn
0x54	0x005408A9	DPSonon
0x55	DSTINVERT 0x00550009	Dn
0x56	0x005601A9	DPSox
0x57	0x00570389	DPSoan
0x58	0x00580785	PDSPoax
0x59	0x00590609	DPSnox
0x5A	PATINVERT 0x005A0049	DPx
0x5B	0x005B18A9	DPSDonox
0x5C	0x005C0649	DPSDxox
0x5D	0x005D0E29	DPSnoan
0x5E	0x005E1B29	DPSDnaox

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x5F	0x005F00E9	DPan
0x60	0x00600365	PDSxa
0x61	0x006116C6	DSPDSaoxxn
0x62	0x00620786	DSPDoax
0x63	0x00630608	SDPnox
0x64	0x00640788	SDPSoax
0x65	0x00650606	DSPnox
0x66	SRCINVERT 0x00660046	DSx
0x67	0x006718A8	SDPSonox
0x68	0x006858A6	DSPDSonoxxn
0x69	0x00690145	PDSxxn
0x6A	0x006A01E9	DPSax
0x6B	0x006B178A	PSDPSoaxxn
0x6C	0x006C01E8	SDPax
0x6D	0x006D1785	PDSPDoaxxn
0x6E	0x006E1E28	SDPSnoax
0x6F	0x006F0C65	PDSxnan
0x70	0x00700CC5	PDSana
0x71	0x00711D5C	SSDxPDxaxn
0x72	0x00720648	SDPSxox
0x73	0x00730E28	SDPnoan
0x74	0x00740646	DSPDxox
0x75	0x00750E26	DSPnoan
0x76	0x00761B28	SDPSnaox
0x77	0x007700E6	DSan
0x78	0x007801E5	PDSax
0x79	0x00791786	DSPDSoaxxn
0x7A	0x007A1E29	DPSDnoax
0x7B	0x007B0C68	SDPxnan

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x7C	0x007C1E24	SPDSnoax
0x7D	0x007D0C69	DPSxnan
0x7E	0x007E0955	SPxDSxo
0x7F	0x007F03C9	DPSaan
0x80	0x008003E9	DPSaa
0x81	0x00810975	SPxDSxon
0x82	0x00820C49	DPSxna
0x83	0x00831E04	SPDSnoaxn
0x84	0x00840C48	SDPxna
0x85	0x00851E05	PDSPnoaxn
0x86	0x008617A6	DSPDSoaxx
0x87	0x008701C5	PDSaxn
0x88	SRCAND 0x008800C6	DSa
0x89	0x00891B08	SDPSnaoxn
0x8A	0x008A0E06	DSPnoa
0x8B	0x008B0666	DSPDxoxn
0x8C	0x008C0E08	SDPnoa
0x8D	0x008D0668	SDPSxoxn
0x8E	0x008E1D7C	SSDxPDxax
0x8F	0x008F0CE5	PDSanan
0x90	0x00900C45	PDSxna
0x91	0x00911E08	SDPSnoaxn
0x92	0x009217A9	DPSDPoaxx
0x93	0x009301C4	SPDaxn
0x94	0x009417AA	PSDPSoaxx
0x95	0x009501C9	DPSaxn
0x96	0x00960169	DPSxx
0x97	0x0097588A	PSDPSonoxx
0x98	0x00981888	SDPSonoxn

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0x99	0x00990066	DSxn
0x9A	0x009A0709	DPSnax
0x9B	0x009B07A8	SDPSoaxn
0x9C	0x009C0704	SPDnax
0x9D	0x009D07A6	DSPDoaxn
0x9E	0x009E16E6	DSPDSaoxx
0x9F	0x009F0345	PDSxan
0xA0	0x00A000C9	DPa
0xA1	0x00A11B05	PDSPnaoxn
0xA2	0x00A20E09	DPSnoa
0xA3	0x00A30669	DPSDxoxn
0xA4	0x00A41885	PDSPonoxn
0xA5	0x00A50065	PDxn
0xA6	0x00A60706	DSPnax
0xA7	0x00A707A5	PDSPoaxn
0xA8	0x00A803A9	DPSoa
0xA9	0x00A90189	DPSoxn
0xAA	0x00AA0029	D
0xAB	0x00AB0889	DPSono
0xAC	0x00AC0744	SPDSxax
0xAD	0x00AD06E9	DPSDaoxn
0xAE	0x00AE0B06	DSPnao
0xAF	0x00AF0229	DPno
0xB0	0x00B00E05	PDSnoa
0xB1	0x00B10665	PDSPxoxn
0xB2	0x00B21974	SSPxDSxox
0xB3	0x00B30CE8	SDPanan
0xB4	0x00B4070A	PSDnax
0xB5	0x00B507A9	DPSDoaxn

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0xB6	0x00B616E9	DPSDPaoxx
0xB7	0x00B70348	SDPxan
0xB8	0x00B8074A	PSDPxax
0xB9	0x00B906E6	DSPDaoxn
0xBA	0x00BA0B09	DPSnao
0xBB	MERGEPAINT 0x00BB0226	DSno
0xBC	0x00BC1CE4	SPDSanax
0xBD	0x00BD0D7D	SDxPDxan
0xBE	0x00BE0269	DPSxo
0xBF	0x00BF08C9	DPSano
0xC0	MERGECOPY 0x00C000CA	PSa
0xC1	0x00C11B04	SPDSnaoxn
0xC2	0x00C21884	SPDSonoxn
0xC3	0x00C3006A	PSxn
0xC4	0x00C40E04	SPDnoa
0xC5	0x00C50664	SPDSxoxn
0xC6	0x00C60708	SDPnax
0xC7	0x00C707AA	PSDPoaxn
0xC8	0x00C803A8	SDPoa
0xC9	0x00C90184	SPDoxn
0xCA	0x00CA0749	DPSDxax
0xCB	0x00CB06E4	SPDSaoxn
0xCC	SRCCOPY 0x00CC0020	S
0xCD	0x00CD0888	SDPono
0xCE	0x00CE0B08	SDPnao
0xCF	0x00CF0224	SPno
0xD0	0x00D00E0A	PSDnoa

Value	Raster-operation code	Boolean function in Reverse Polish Notation
0xD1	0x00D1066A	PSDPxoxn
0xD2	0x00D20705	PDSnax
0xD3	0x00D307A4	SPDSoaxn
0xD4	0x00D41D78	SSPxPDxax
0xD5	0x00D50CE9	DPSanan
0xD6	0x00D616EA	PSDPSaoxx
0xD7	0x00D70349	DPSxan
0xD8	0x00D80745	PDSPxax
0xD9	0x00D906E8	SDPSaoxn
0xDA	0x00DA1CE9	DPSDanax
0xDB	0x00DB0D75	SPxDSxan
0xDC	0x00DC0B04	SPDnao
0xDD	0x00DD0228	SDno
0xDE	0x00DE0268	SDPxo
0xDF	0x00DF08C8	SDPano
0xE0	0x00E003A5	PDSoa
0xE1	0x00E10185	PDSoxn
0xE2	0x00E20746	DSPDxax
0xE3	0x00E306EA	PSDPaoxn
0xE4	0x00E40748	SDPSxax
0xE5	0x00E506E5	PDSPaoxn
0xE6	0x00E61CE8	SDPSanax
0xE7	0x00E70D79	SPxPDxan
0xE8	0x00E81D74	SSPxDSxax
0xE9	0x00E95CE6	DSPDSanaxxn
0xEA	0x00EA02E9	DPSao
0xEB	0x00EB0849	DPSxno
0xEC	0x00EC02E8	SDPao
0xED	0x00ED0848	SDPxno

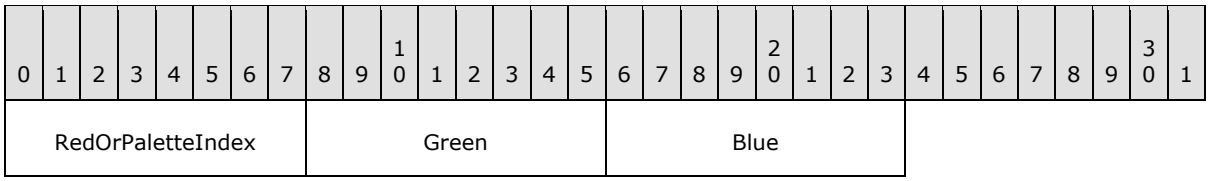
Value	Raster-operation code	Boolean function in Reverse Polish Notation
0xEE	SRCPAINT 0x00EE0086	DSo
0xEF	0x00EF0A08	SDPnoo
0xF0	PATCOPY 0x00F00021	P
0xF1	0x00F10885	PDSono
0xF2	0x00F20B05	PDSnao
0xF3	0x00F3022A	PSno
0xF4	0x00F40B0A	PSDnao
0xF5	0x00F50225	PDno
0xF6	0x00F60265	PDSxo
0xF7	0x00F708C5	PDSano
0xF8	0x00F802E5	PDSao
0xF9	0x00F90845	PDSxno
0xFA	0x00FA0089	DPo
0xFB	PATPAINT 0x00FB0A09	DPSnoo
0xFC	0x00FC008A	PSo
0xFD	0x00FD0A0A	PSDnoo
0xFE	0x00FE02A9	DPSoo
0xFF	WHITENESS 0x00FF0062	1

For example, by using the previous table, one can derive that the raster operation at index 0xEE (SRCPAINT) replaces the values of the pixels in the destination bitmap with a combination of pixel values of the destination and source bitmaps.

For more information about ternary raster operations, see [MSDN-TRO] and [WGFX] section 11.1.

2.2.2.3.1.1.1.8 Generic Color (TS_COLOR)

The TS_COLOR structure holds a 3-byte RGB color triplet (the red, green, and blue components necessary to reproduce a color in the additive RGB space) or a 1-byte palette index.



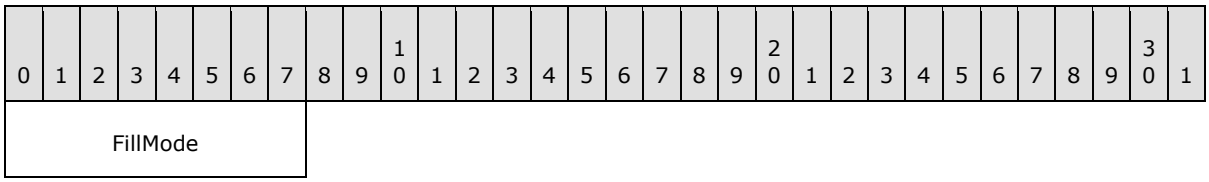
RedOrPaletteIndex (1 byte): An 8-bit unsigned integer. **RedOrPaletteIndex** is used as a palette index for 16-color and 256-color palletized color schemes. If the RGB color scheme is in effect, this field contains the red RGB component.

Green (1 byte): An 8-bit unsigned integer. **Green** contains the green RGB color component.

Blue (1 byte): An 8-bit unsigned integer. **Blue** contains the blue RGB color component.

2.2.2.3.1.1.1.9 Fill Mode (FILL_MODE)

The FILL_MODE structure specifies the algorithm used to determine if a given point is contained within a polygon.



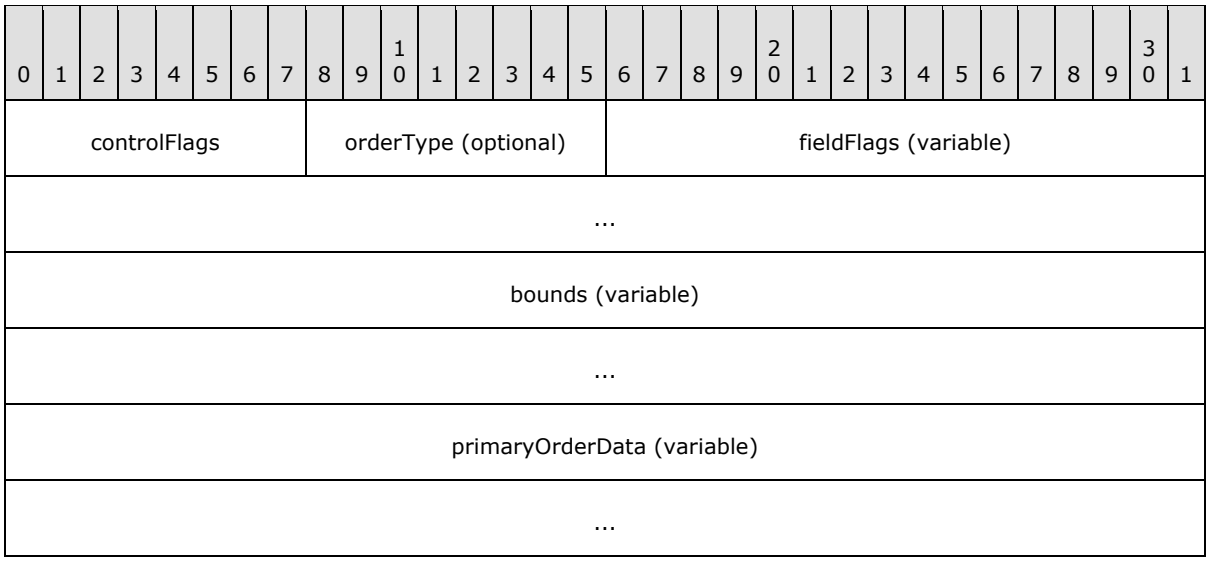
FillMode (1 byte): An 8-bit unsigned integer that specifies the fill mode. **FillMode** MUST be one of the following values.

Value	Meaning
ALTERNATE 0x00	Alternate fill mode. To test if a point is inside a polygon, a ray is drawn from that point parallel to the x-axis to infinity, and the number of intersections between the ray and the polygon outline is counted. If the number of intersections is odd, the point is inside the polygon, and, if it is even, the point is outside.
WINDING 0x01	Winding fill mode. To determine if a point is within a polygon, a ray is drawn from that point parallel to the x-axis to infinity, and its intersections with the outline are examined. A count that starts from 0 is kept for the ray. Each intersection with a clockwise polygon outline increments the count, and each intersection with a counterclockwise polygon outline decrements the count. If the final value of the count is nonzero, the point is inside the polygon; otherwise, it is outside.

For alternate and winding fill modes, see [WGFX] section 9.4 and [PW] pages 169–171.

2.2.2.3.1.1.2 Primary Drawing Order (PRIMARY_DRAWING_ORDER)

The PRIMARY_DRAWING_ORDER structure encapsulates a primary drawing order. All primary drawing orders employ special field encoding to reduce the number of bytes sent on the wire. Field encoding maintains a copy of the most recent field values that were used in each primary drawing order, in addition to common state information such as the last bounding rectangle used across all orders and the last order type. Only the fields that have changed are ever sent on the wire. This implies that all the fields in a primary drawing order are optional, their presence being denoted by the **controlFlags** and **fieldFlags** fields.



controlFlags (1 byte): An 8-bit unsigned integer. A control byte that identifies the class of the drawing order, and describes the fields that are included in the order and the type of coordinates being used. This field MUST contain a combination of the following flags.

Value	Meaning
TS_STANDARD 0x01	Indicates that the order is a primary drawing order. This flag MUST be set.
TS_BOUNDS 0x04	Indicates that the order has a bounding rectangle.
TS_TYPE_CHANGE 0x08	Indicates that the order type has changed and that the **orderType** field is present.
TS_DELTA_COORDINATES 0x10	Indicates that all of the Coord-type fields in the order (see section 2.2.2.3.1.1.1.1) are specified as one-byte signed deltas from their previous values.
TS_ZERO_BOUNDS_DELTAS 0x20	Indicates that the previous bounding rectangle MUST be used, as the bounds have not changed (this implies that the bounds field is not present).
TS_ZERO_FIELD_BYTE_BIT0 0x40	Used in conjunction with the TS_ZERO_FIELD_BYTE_BIT1 (0x80) flag to form a two-bit count (so maximum of 3) of the number of field flag bytes (present in the **fieldFlags** field) that are zero and not present, counted from the end of the set of field flag bytes. This flag is the least significant bit of the count.
TS_ZERO_FIELD_BYTE_BIT1 0x80	Used in conjunction with the TS_ZERO_FIELD_BYTE_BIT0 (0x40) flag to form a two-bit count (so maximum of 3) of the number of field flag bytes (present in the **fieldFlags** field) that are zero and not present, counted from the end of the set of field flag bytes. This flag is the most significant bit of the count.

For the usage of the TS_ZERO_FIELD_BYTE_BIT0 (0x40) and TS_ZERO_FIELD_BYTE_BIT1 (0x80) flags, see the **fieldFlags** field below.

orderType (1 byte): An 8-bit unsigned integer. An optional identifier describing the type of primary drawing order. The initial value for the **orderType** agreed on by both the server and client is TS_ENC_PATBLT_ORDER (0x01) for the PatBlt primary drawing order.

Value	Meaning
TS_ENC_DSTBLT_ORDER 0x00	DstBlt (section 2.2.2.3.1.1.2.1) Primary Drawing Order.
TS_ENC_PATBLT_ORDER 0x01	PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.
TS_ENC_SCRBLT_ORDER 0x02	ScrBlt (section 2.2.2.3.1.1.2.7) Primary Drawing Order.
TS_ENC_DRAWNINEGRID_ORDER 0x07	DrawNineGrid (section 2.2.2.3.1.1.2.21) Primary Drawing Order.
TS_ENC_MULTI_DRAWNINEGRID_ORDER 0x08	MultiDrawNineGrid (section 2.2.2.3.1.1.2.22) Primary Drawing Order.
TS_ENC_LINETO_ORDER 0x09	LineTo (section 2.2.2.3.1.1.2.11) Primary Drawing Order.
TS_ENC_OPAQUERECT_ORDER 0x0A	OpaqueRect (section 2.2.2.3.1.1.2.5) Primary Drawing Order.
TS_ENC_SAVEBITMAP_ORDER 0x0B	SaveBitmap (section 2.2.2.3.1.1.2.12) Primary Drawing Order.
TS_ENC_MEMBLT_ORDER 0x0D	MemBlt (section 2.2.2.3.1.1.2.9) Primary Drawing Order.
TS_ENC_MEM3BLT_ORDER 0x0E	Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Order.
TS_ENC_MULTIDSTBLT_ORDER 0x0F	MultiDstBlt (section 2.2.2.3.1.1.2.2) Primary Drawing Order.
TS_ENC_MULTIPATBLT_ORDER 0x10	MultiPatBlt (section 2.2.2.3.1.1.2.4) Primary Drawing Order.
TS_ENC_MULTIOPAQUERECT_ORDER 0x12	MultiOpaqueRect (section 2.2.2.3.1.1.2.6) Primary Drawing Order.
TS_ENC_MULTISCRBLT_ORDER 0x11	MultiScrBlt (section 2.2.2.3.1.1.2.8) Primary Drawing Order.
TS_ENC_FAST_INDEX_ORDER 0x13	FastIndex (section 2.2.2.3.1.1.2.14) Primary Drawing Order.
TS_ENC_POLYGON_SC_ORDER 0x14	PolygonSC (section 2.2.2.3.1.1.2.16) Primary Drawing Order.
TS_ENC_POLYGON_CB_ORDER 0x15	PolygonCB (section 2.2.2.3.1.1.2.17) Primary Drawing Order.
TS_ENC_POLYLINE_ORDER 0x16	Polyline (section 2.2.2.3.1.1.2.18) Primary Drawing Order.

Value	Meaning
TS_ENC_FAST_GLYPH_ORDER 0x18	FastGlyph (section 2.2.2.3.1.1.2.15) Primary Drawing Order.
TS_ENC_ELLIPSE_SC_ORDER 0x19	EllipseSC (section 2.2.2.3.1.1.2.19) Primary Drawing Order.
TS_ENC_ELLIPSE_CB_ORDER 0x1A	EllipseCB (section 2.2.2.3.1.1.2.20) Primary Drawing Order.
TS_ENC_INDEX_ORDER 0x1B	GlyphIndex (section 2.2.2.3.1.1.2.13) Primary Drawing Order.

fieldFlags (variable): A variable-length 1-byte to 3-byte field. The optional **fieldFlags** field is used to indicate the presence of an order field in the encoded fields portion of the packet (represented by the **primaryOrderData** field). Each bit in the **fieldFlags** field functions as a flag and indicates if a particular order field is present.

The number of bytes used to represent the field flags will vary based on the maximum number of fields a primary drawing order can hold. For any given drawing order, the size (in bytes) of the **fieldFlags** field can be computed with the following formula:

```
ceil(((numberOfOrderFields) + 1) / 8)
```

The maximum number of allowed fields is 23, which corresponds to three field flags bytes. The "+ 1" in the equation indicates that for seven fields there is one field flags byte; however, for eight fields, there are two bytes. (This drives some of the specialized orders in RDP where a special case may be used to reduce the number of fields below 7 or 15, thereby reducing the number of field flags bytes sent.)

The individual field flags bytes are ordered as a little-endian DWORD; the low order byte is first. The order of the flags proceeds from 0x01 (the least significant bit), which corresponds to the first field in the order, 0x02 the second field in the order, 0x04 the third field in the order, and so on.

The presence of the **fieldFlags** field is also governed by the **controlFlags** field ZERO_FIELD_BYTE_BIT0 (0x40) and ZERO_FIELD_BYTE_BIT1 (0x80) flags (known as the "zero flags"). These flags are used as a two-bit count of the number of field flags bytes that are zero and not present, counted from the end of the set of flag bytes.

Zero Flags		Actual number of field flag bytes sent on wire		
Bit 1	Bit 0	Orders with 1 field-encoding byte	Orders with 2 field-encoding bytes	Orders with 3 field-encoding bytes
0	0	1	2	3
0	1	0	1	2
1	0	N/A	0	1
1	1	N/A	N/A	0

Figure 2: fieldFlags diagram

For example, assume that a given order has nine fields. Hence, there can be up to two bytes in the fieldFlags field. Assume further that the fieldFlags field has the field-encoding bytes {0x20, 0x00}. In this case, there is one zero byte (counting from the end backward); thus, the **controlFlags** field contains the ZERO_FIELD_BYTE_BIT0 flag (0x40), and the order is sent on the wire with a **fieldFlags** value of 0x20.

bounds (variable): A variable-length 1-byte to 9-byte field. The presence of the optional **bounds** field is governed by the TS_BOUNDS (0x04) flag in the **controlFlags** field, which indicates that the order must have a bounding region applied. If the **controlFlags** field TS_ZERO_BOUNDS_DELTAS (0x20) flag is also set, the previous bounding rectangle MUST be used, as the bounds have not changed (this implies that the bounds field is not present). Otherwise, the bounds are encoded as an encoding description byte followed by one or more encoded bounds. The description byte contains two flags for each of the following left, top, right, and bottom rectangle components.

Flag	Meaning
TS_BOUND_LEFT 0x01	Indicates that the left bound is present and encoded as a 2-byte little-endian ordered value.
TS_BOUND_TOP 0x02	Indicates that the top bound is present and encoded as a 2-byte little-endian ordered value.
TS_BOUND_RIGHT 0x04	Indicates that the right bound is present and encoded as a 2-byte little-endian ordered value.
TS_BOUND_BOTTOM 0x08	Indicates that the bottom bound is present and encoded as a 2-byte little-endian ordered value.
TS_BOUND_DELTA_LEFT 0x10	Indicates that the left bound is present and encoded as a one-byte signed value used as an offset (-128 to 127) from the previous value.
TS_BOUND_DELTA_TOP 0x20	Indicates that the top bound is present and encoded as a one-byte signed value used as an offset (-128 to 127) from the previous value.
TS_BOUND_DELTA_RIGHT 0x40	Indicates that the right bound is present and encoded as a one-byte signed value used as an offset (-128 to 127) from the previous value.
TS_BOUND_DELTA_BOTTOM 0x80	Indicates that the bottom bound is present and encoded as a one-byte signed value used as an offset (-128 to 127) from the previous value.

If for a given component a TS_BOUND_XXX or TS_BOUND_DELTA_XXX flag is not present, the component value is the same as the last one used, and no value is included.

The initial value for the bounds agreed on by both the server and client is the zero rectangle:

```
(left, top, right, bottom) = (0, 0, 0, 0)
```

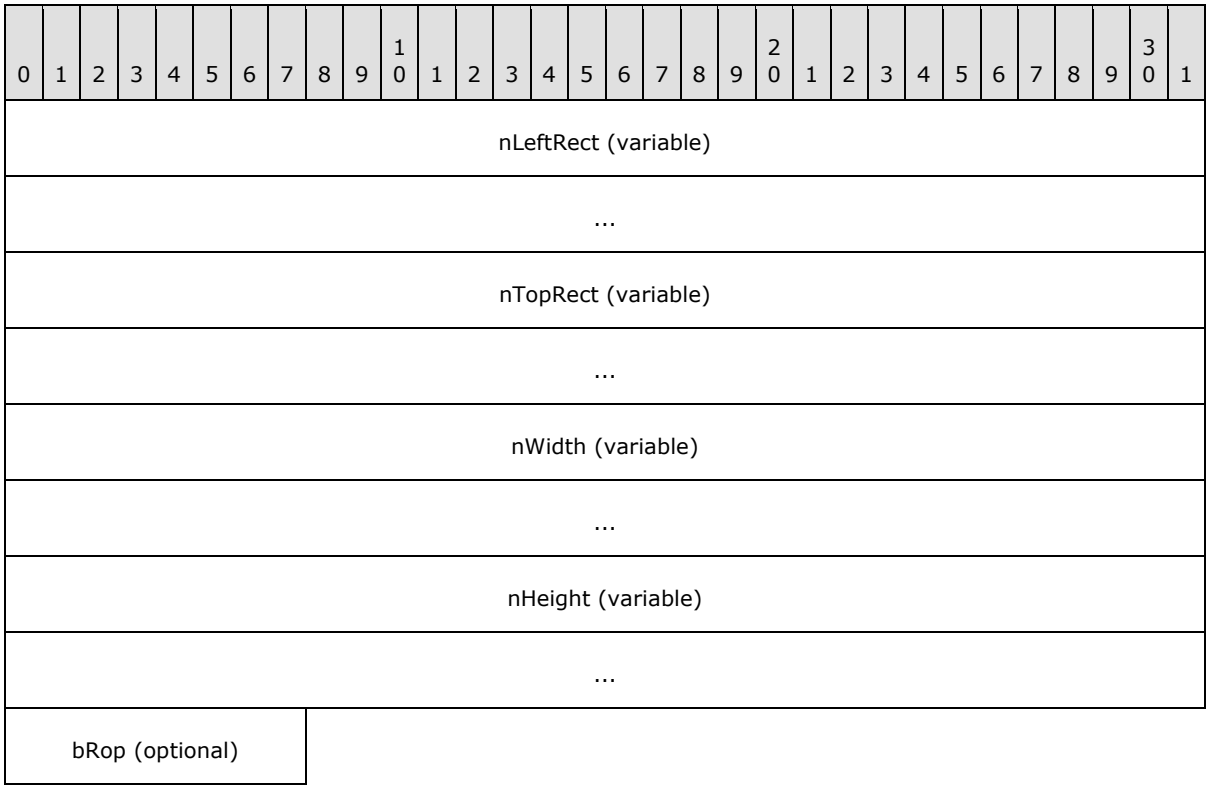
primaryOrderData (variable): A variable-length byte field. The **primaryOrderData** field contains the encoded order field values whose presence is governed by the **fieldFlags** field. The encoded fields, if present, MUST be encoded in the same order as the order description

(see sections 2.2.2.3.1.1.2.1 through 2.2.2.3.1.1.2.22). If a field is not present, its value is the same as the last value sent for that order. The initial values for all fields are zero.

2.2.2.3.1.1.2.1 DstBlt (DSTBLT_ORDER)

The DstBlt Primary Drawing Order is used to paint a rectangle using a destination-only raster operation.

```
Encoding order number: 0 (0x00)
Negotiation order number: 0 (0x00)
Number of fields: 5
Number of field encoding bytes: 1
Maximum encoded field length: 9 bytes
```



nLeftRect (variable): Left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): Top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): Width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

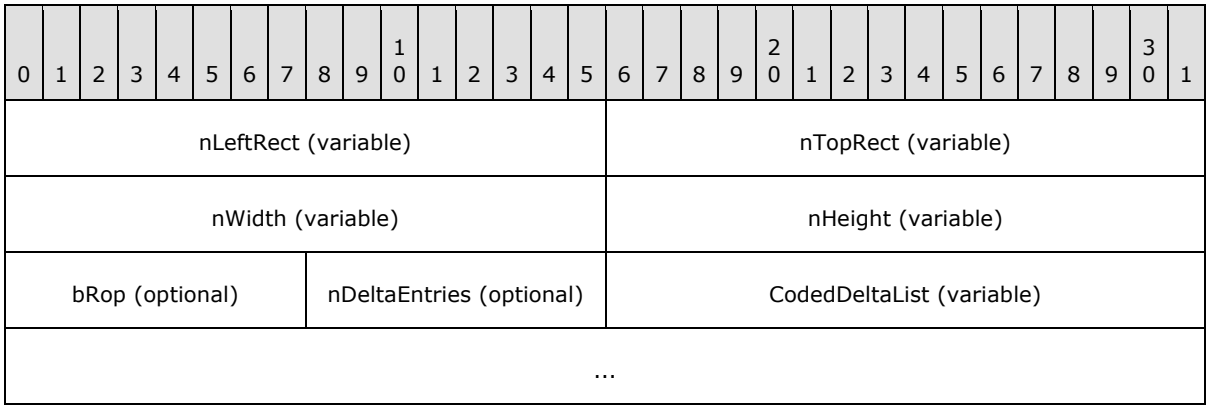
nHeight (variable): Height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): Index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7). The resultant ROP3 operation MUST only depend on the destination bits (there MUST NOT be any dependence on source or pattern bits).

2.2.2.3.1.1.2.2 MultiDstBlt (MULTI_DSTBLT_ORDER)

The MultiDstBlt Primary Drawing Order is used to paint multiple rectangles using a destination-only raster operation.

```
Encoding order number: 15 (0x0F)
Negotiation order number: 15 (0x0F)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 395 bytes
```



nLeftRect (2 bytes): Signed one-byte or two-byte field. Left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (2 bytes): Signed one-byte or two-byte field. Top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (2 bytes): Signed one-byte or two-byte field. Width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (2 bytes): Signed one-byte or two-byte field. Height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): Index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7). The resultant ROP3 operation MUST only depend on the destination bits (there MUST NOT be any dependence on source or pattern bits).

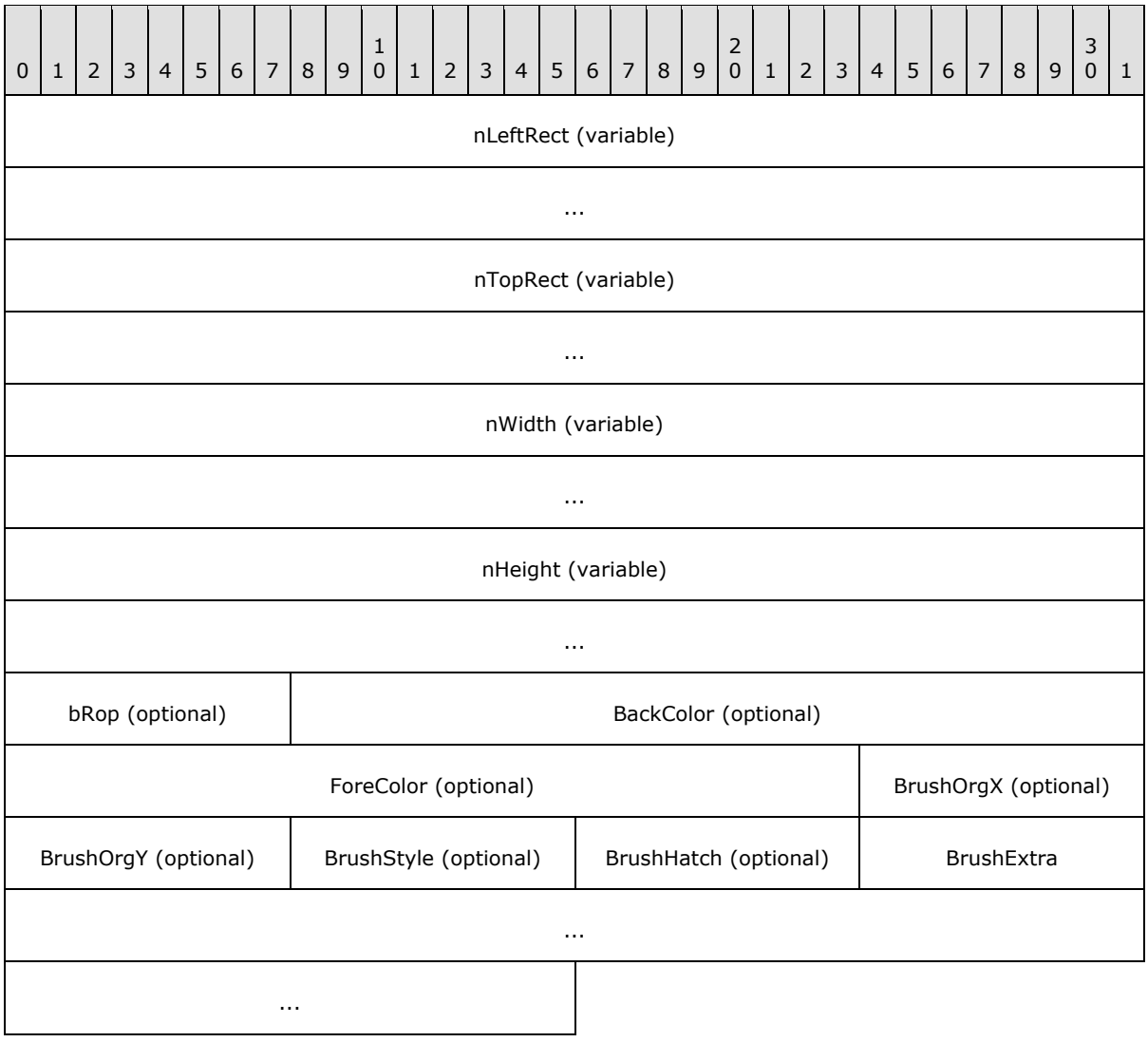
nDeltaEntries (1 byte): An 8-bit unsigned integer. The number of bounding rectangles described by the **CodedDeltaList** field.

CodedDeltaList (variable): A Two-Byte Header Variable Field (section 2.2.2.3.1.1.1.3) structure that encapsulates a Delta-Encoded Rectangles (section 2.2.2.3.1.1.1.5) structure that contains bounding rectangles to use when rendering the order. The number of rectangles described by the Delta-Encoded Rectangles structure is specified by the **nDeltaEntries** field.

2.2.2.3.1.1.2.3 PatBlt (PATBLT_ORDER)

The PatBlt Primary Drawing Order is used to paint a rectangle using a specified brush and 3-way raster operation.

```
Encoding order number: 1 (0x01)
Negotiation order number: 1 (0x01)
Number of fields: 12
Number of field encoding bytes: 2
Maximum encoded field length: 26 bytes
```



nLeftRect (variable): Left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): Top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): Width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): Height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): Index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7).

BackColor (3 bytes): Background color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): Foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BrushOrgX (1 byte): An 8-bit signed integer. The X coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushOrgY (1 byte): An 8-bit signed integer. The Y coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushStyle (1 byte): An 8-bit unsigned integer. The style of the brush used in the drawing operation.

If the TS_CACHED_BRUSH (0x80) flag is set in the most significant bit of the BrushStyle field, a brush that was previously cached using the Cache Bitmap Secondary Order (see section 2.2.2.3.1.2.7) MUST be used. In this case, the BrushHatch field MUST contain the index of the Brush Cache entry that holds the selected brush to use, while the low nibble of the BrushStyle field MUST contain an identifier describing the color depth of the cached brush.

Value	Meaning
BMF_1BPP 0x1	1 bit-per-pixel
BMF_8BPP 0x3	8 bits-per-pixel
BMF_16BPP 0x4	15 or 16 bits-per-pixel
BMF_24BPP 0x5	24 bits-per-pixel

If the TS_CACHED_BRUSH (0x80) flag is not set in the most significant bit of the BrushStyle field, an 8 pixel by 8 pixel brush MUST be used, and one of the following style identifiers MUST be present in the field.

Value	Meaning
BS_SOLID 0x00	Solid color brush. The BrushHatch field SHOULD be set to 0.
BS_NULL 0x01	Hollow brush. The BrushHatch field SHOULD be set to 0.

Value	Meaning
BS_HATCHED 0x02	Hatched brush. The hatch pattern MUST be described by the BrushHatch field (there are six possible hatch patterns).
BS_PATTERN 0x03	Pattern brush. The pixel pattern MUST be described by the BrushExtra and BrushHatch fields.

BrushHatch (1 byte): An 8-bit unsigned integer. Holds a brush hatch identifier or a Brush Cache index, depending on the contents of the BrushStyle field.

If the TS_CACHED_BRUSH (0x80) flag is set in the most significant bit of the BrushStyle field, the BrushHatch value MUST contain the index of the Brush Cache entry that holds the selected brush to use.

If the BrushStyle field is set to BS_SOLID (0x00) or BS_NULL (0x01), the BrushHatch field SHOULD be set to 0.

If the BrushStyle field is set to BS_HATCHED (0x02), the BrushHatch field MUST contain one of the following hatch constants:

Value	Meaning
HS_HORIZONTAL 0x00	
HS_VERTICAL 0x01	
HS_FDIAGONAL 0x02	
HS_BDIAGONAL 0x03	
HS_CROSS 0x04	
HS_DIAGCROSS 0x05	

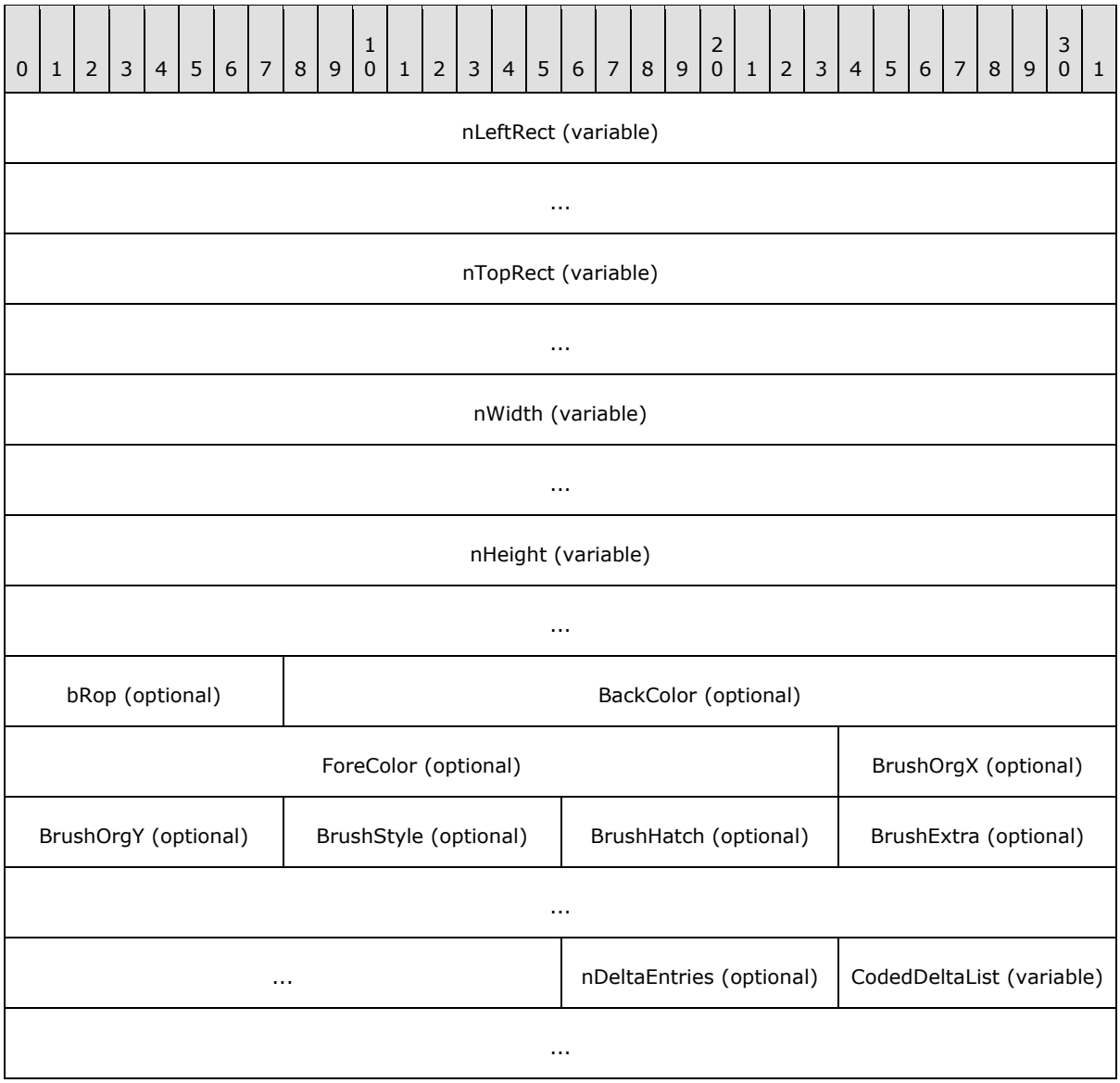
If the **BrushStyle** field is set to BS_PATTERN (0x03), the **BrushHatch** field MUST encode the pixel pattern present in the bottom row of the 8x8 pattern brush (the pixel patterns in the top seven rows MUST be encoded in the **BrushExtra** field). For example, if the bottom row of the pattern brush contains an alternating series of black and white pixels, **BrushHatch** will contain either 0xAA or 0x55.

BrushExtra (7 bytes): A 7-byte, byte array. **BrushExtra** contains an array of bitmap bits that encodes the pixel pattern present in the top seven rows of the 8x8 pattern brush. The pixel pattern present in the bottom row is encoded in the **BrushHatch** field. The **BrushExtra** field is only present if the **BrushStyle** is set to BS_PATTERN (0x03). The rows are encoded in reverse order, that is, the pixels in the penultimate row are encoded in the first byte, and the pixels in the top row are encoded in the seventh byte. For example, a 45-degree downward sloping left-to-right line would be encoded in **BrushExtra** as { 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 } with **BrushHatch** containing the value 0x01 (the bottom row).

2.2.2.3.1.1.2.4 MultiPatBlt (MULTI_PATBLT_ORDER)

The MultiPatBlt Primary Drawing Order is used to paint multiple rectangles using a specified brush and 3-way raster operation.

```
Encoding order number: 16 (0x10)
Negotiation order number: 16 (0x10)
Number of fields: 14
Number of field encoding bytes: 2
Maximum encoded field length: 412 bytes
```



nLeftRect (variable): The left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): The width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): The height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): The index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7).

BackColor (3 bytes): The background color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): The foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BrushOrgX (1 byte): An 8-bit signed integer. The X coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushOrgY (1 byte): An 8-bit signed integer. The Y coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushStyle (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushStyle** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushHatch (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushHatch** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushExtra (7 bytes): The contents and format of this field are the same as the **BrushExtra** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

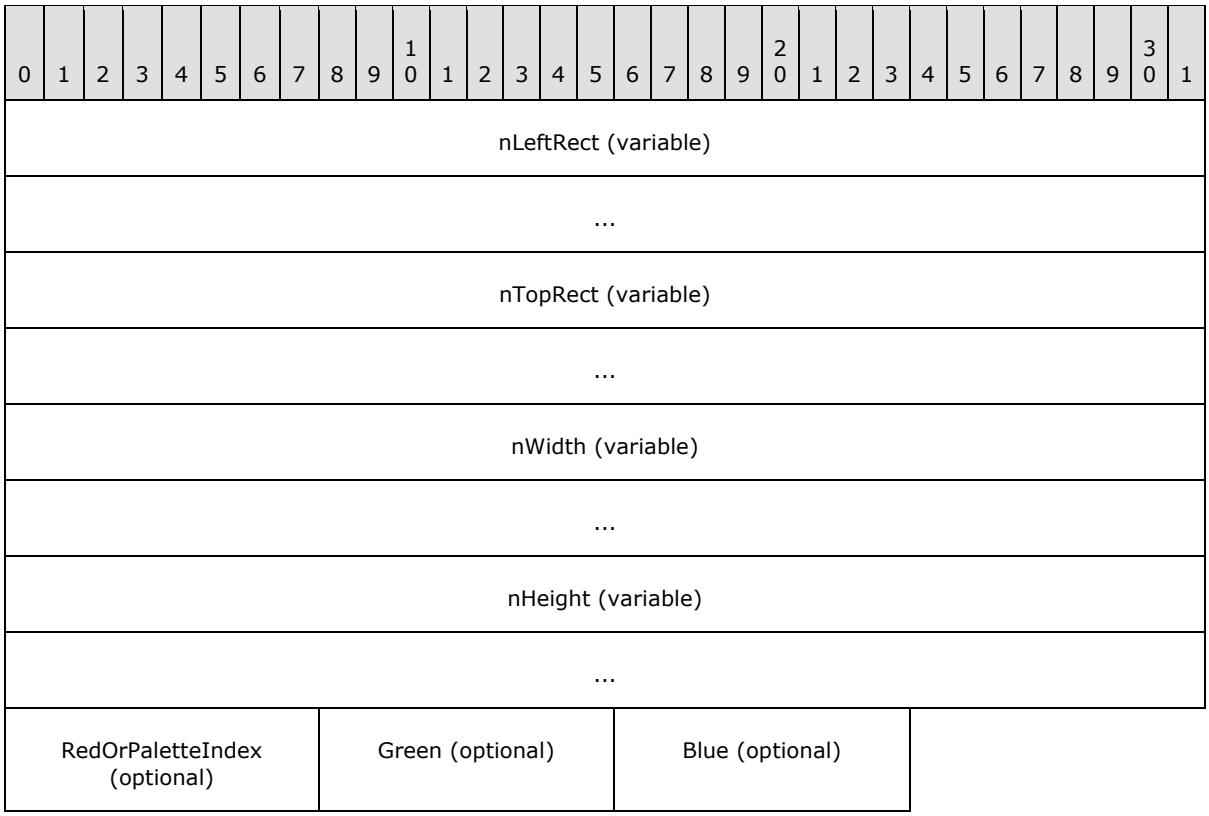
nDeltaEntries (1 byte): An 8-bit unsigned integer. The number of bounding rectangles described by the **CodedDeltaList** field.

CodedDeltaList (variable): A Two-Byte Header Variable Field (section 2.2.2.3.1.1.1.3) structure that encapsulates a Delta-Encoded Rectangles (section 2.2.2.3.1.1.1.5) structure that contains bounding rectangles to use when rendering the order. The number of rectangles described by the Delta-Encoded Rectangles structure is specified by the **nDeltaEntries** field.

2.2.2.3.1.1.2.5 OpaqueRect (OPAQUERECT_ORDER)

The OpaqueRect Primary Drawing Order is used to paint a rectangle using an opaque brush.

```
Encoding order number: 10 (0x0A)
Negotiation order number: 10 (0x0A)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 11 bytes
```



nLeftRect (variable): The left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): The width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): The height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

RedOrPaletteIndex (1 byte): An 8-bit unsigned integer. Used as a palette index for 16-color and 256-color palletized color schemes. If the RGB color scheme is in effect, this field holds the red RGB component.

Green (1 byte): An 8-bit unsigned integer. The green RGB color component.

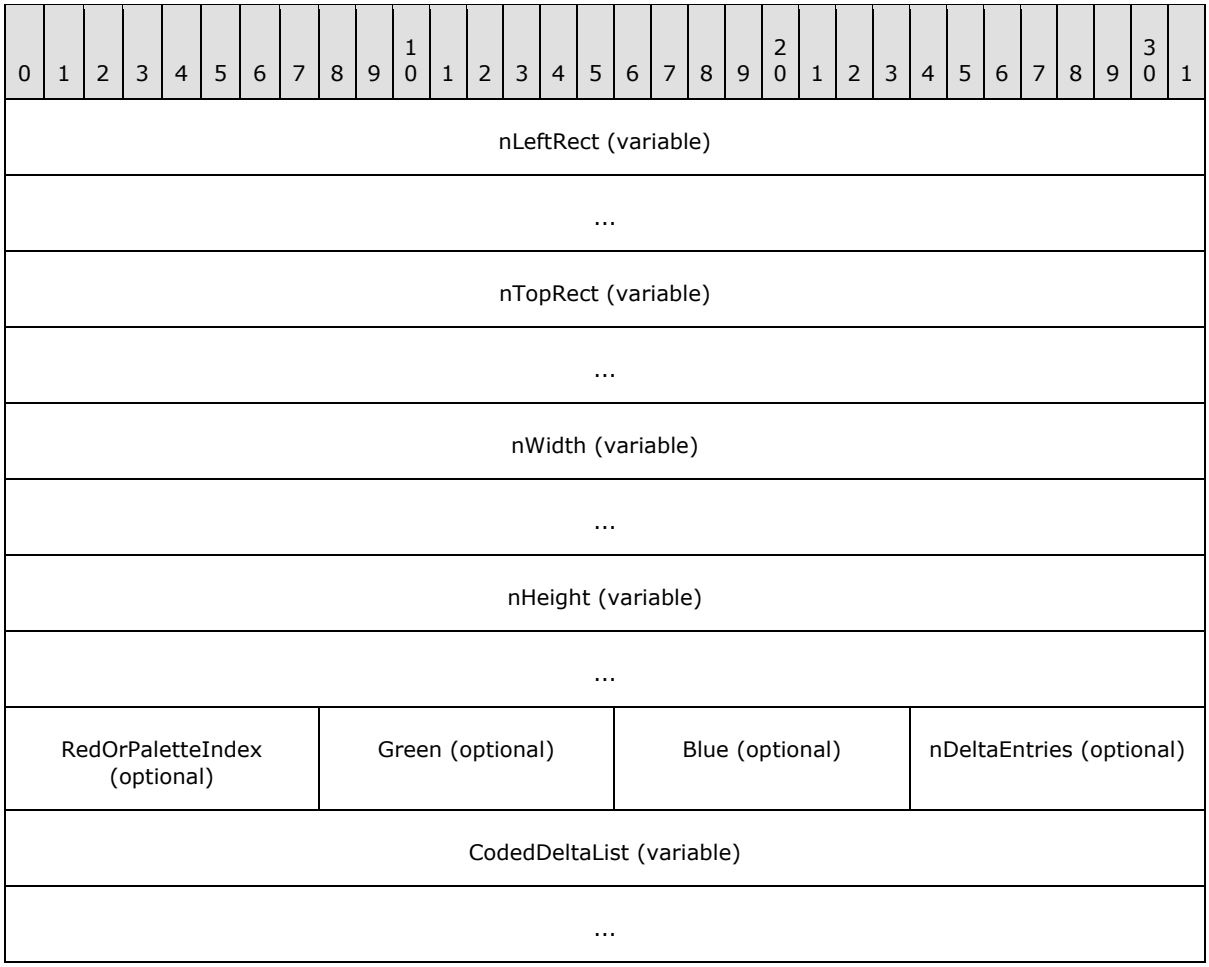
Blue (1 byte): An 8-bit unsigned integer. The blue RGB color component.

2.2.2.3.1.1.2.6 MultiOpaqueRect (MULTI_OPAQUERECT_ORDER)

The MultiOpaqueRect Primary Drawing Order is used to paint multiple rectangles using an opaque brush.

```
Encoding order number: 18 (0x12)
Negotiation order number: 18 (0x12)
```

```
Number of fields: 9
Number of field encoding bytes: 2
Maximum encoded field length: 397 bytes
```



nLeftRect (variable): The left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): The width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): The height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

RedOrPaletteIndex (1 byte): An 8-bit unsigned integer. Used as a palette index for 16-color and 256-color palletized color schemes. If the RGB color scheme is in effect, holds the red RGB component.

Green (1 byte): An 8-bit unsigned integer. The green RGB color component.

Blue (1 byte): An 8-bit unsigned integer. The blue RGB color component.

nDeltaEntries (1 byte): An 8-bit unsigned integer. The number of bounding rectangles described by the **CodedDeltaList** field.

CodedDeltaList (variable): A Two-Byte Header Variable Field (section 2.2.2.3.1.1.1.3) structure that encapsulates a Delta-Encoded Rectangles (section 2.2.2.3.1.1.1.5) structure that contains bounding rectangles to use when rendering the order. The number of rectangles described by the Delta-Encoded Rectangles structure is specified by the **nDeltaEntries** field.

2.2.2.3.1.1.2.7 ScrBlt (SCRBLT_ORDER)

The ScrBlt Primary Drawing Order is used to perform a bit-block transfer from a source region to a destination region of the screen.

Encoding order number: 2 (0x02)

```
Negotiation order number: 2 (0x02)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 13 bytes
```

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
nLeftRect (variable)																															
...																															
nTopRect (variable)																															
...																															
nWidth (variable)																															
...																															
nHeight (variable)																															
...																															
bRop (optional)									nXSrc (variable)																					
...																															
nYSrc (variable)																															
...																															

nLeftRect (variable): The left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): The width of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): The height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): The index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7). The resultant ROP3 operation MUST only depend on the destination and source bits (there MUST NOT be any dependence on pattern bits).

nXSrc (variable): The X coordinate of the source rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nYSrc (variable): The Y coordinate of the source rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

2.2.2.3.1.1.2.8 MultiScrBlt (MULTI_SCRBLT_ORDER)

The MultiScrBlt Primary Drawing Order is used to perform multiple bit-block transfers from source regions to destination regions of the screen.

```
Encoding order number: 17 (0x11)
Negotiation order number: 17 (0x11)
Number of fields: 9
Number of field encoding bytes: 2
Maximum encoded field length: 399 bytes
```

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
nLeftRect (variable)																															
...																															
nTopRect (variable)																															
...																															
nWidth (variable)																															
...																															
nHeight (variable)																															
...																															
bRop (optional)									nXSrc (variable)																					
...																															
nYSrc (variable)																															
...																															
nDeltaEntries (optional)									CodedDeltaList (variable)																					
...																															

nLeftRect (variable): The left coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): The width of the destination rectangle specified using a Coord Field (see Section 2.2.2.3.1.1.1.1).

nHeight (variable): The height of the destination rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): The index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7). The resultant ROP3 operation MUST only depend on the destination and source bits (there MUST NOT be any dependence on pattern bits).

nXSrc (variable): The X coordinate of the source rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nYSrc (variable): The Y coordinate of the source rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

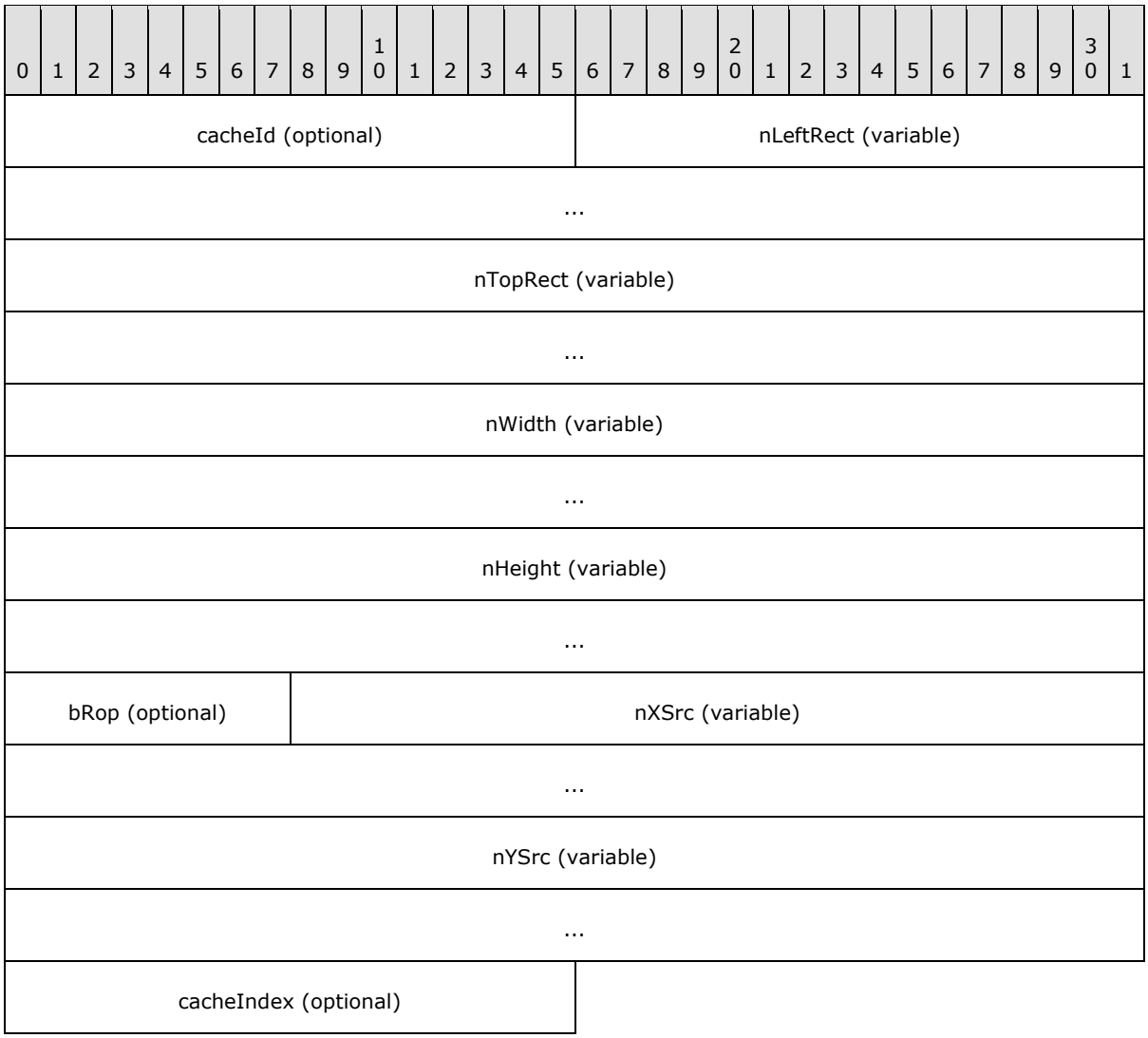
nDeltaEntries (1 byte): An 8-bit unsigned integer. The number of bounding rectangles described by the **CodedDeltaList** field.

CodedDeltaList (variable): A Two-Byte Header Variable Field (section 2.2.2.3.1.1.1.3) structure that encapsulates a Delta-Encoded Rectangles (section 2.2.2.3.1.1.1.5) structure that contains bounding rectangles to use when rendering the order. The number of rectangles described by the Delta-Encoded Rectangles structure is specified by the **nDeltaEntries** field.

2.2.2.3.1.1.2.9 MemBlt (MEMBLT_ORDER)

The MemBlt Primary Drawing Order is used to render a bitmap stored in the bitmap cache or offscreen bitmap cache to the screen.

```
Encoding order number: 13 (0x0D)
Negotiation order number: 3 (0x03)
Number of fields: 9
Number of field encoding bytes: 2
Maximum encoded field length: 17 bytes
```



cacheId (2 bytes): A 16-bit unsigned integer. The **cacheId** field contains the encoded bitmap cache ID and Color Table Cache entry.

The high byte contains the index of the color table entry to use (cached previously with a Cache Color Table (section 2.2.2.3.1.2.4) Secondary Drawing Order) while the low byte contains the ID of the bitmap cache in which the source bitmap is stored (cached previously with a Cache Bitmap - Revision 1 (section 2.2.2.3.1.2.2) or Cache Bitmap - Revision 2 (section 2.2.2.3.1.2.3) Secondary Drawing Order).

The color table entry MUST be in the range 0 to 5 (inclusive), while the bitmap cache ID MUST be in the range negotiated by the Revision 1 or 2 Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5). However, if the bitmap cache ID is set to TS_BITMAPCACHE_SCREEN_ID (0xFF), the cacheIndex field MUST contain the index of an entry in the Offscreen Bitmap Cache that contains the source bitmap.

nLeftRect (variable): Left coordinate of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): Top coordinate of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): Width of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): Height of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): Index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7). The resultant ROP3 operation MUST only depend on the destination and source bits (there MUST NOT be any dependence on pattern bits).

nXSrc (variable): X coordinate of the source rectangle within the source bitmap specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nYSrc (variable): The inverted Y coordinate of the source rectangle within the source bitmap specified using a Coord Field (section 2.2.2.3.1.1.1.1). The actual value of the Y coordinate MUST be computed using the following formula:

```
ActualYSrc = (SourceBitmapHeight - nHeight) - nYSrc
```

cacheIndex (2 bytes): A 16-bit unsigned integer. The index of the source bitmap within the bitmap cache specified by the **cacheId** field. This value MUST be in the range negotiated by the Revision 1 or 2 Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5).

If this field is set to BITMAPCACHE_WAITING_LIST_INDEX (32767), the last bitmap cache entry MUST contain the bitmap.

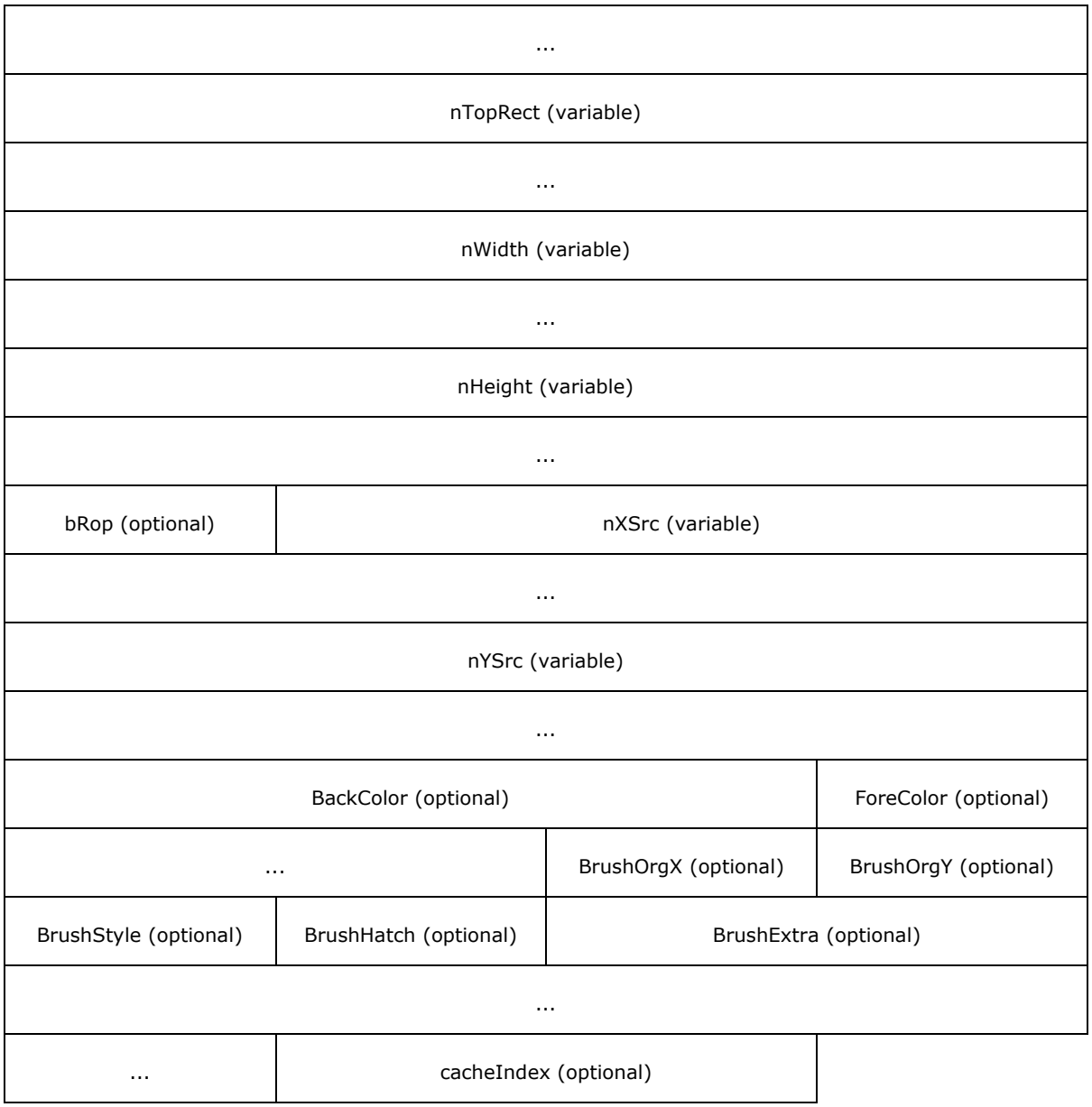
If the bitmap cache ID (specified in the **cacheId** field) is set to TS_BITMAPCACHE_SCREEN_ID (0xFF), this field MUST contain the index of an entry in the Offscreen Bitmap Cache that contains the source bitmap.

2.2.2.3.1.1.2.10 Mem3Blt (MEM3BLT_ORDER)

The Mem3Blt Primary Drawing Order is used to render a bitmap stored in the bitmap cache or offscreen bitmap cache to the screen using a specified brush and 3-way raster operation.

```
Encoding order number: 14 (0x0E)
Negotiation order number: 4 (0x04)
Number of fields: 16
Number of field encoding bytes: 3
Maximum encoded field length: 34 bytes
```

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cacheId (optional)																nLeftRect (variable)															



cacheId (2 bytes): A 16-bit unsigned integer. The **cacheId** field contains the encoded bitmap cache ID and Color Table Cache entry. The high byte contains the index of the color table entry to use (cached previously with a Cache Color Table (section 2.2.2.3.1.2.4) Secondary Drawing Order) while the low byte contains the ID of the bitmap cache in which the source bitmap is stored (cached previously with a Cache Bitmap - Revision 1 (section 2.2.2.3.1.2.2) or Cache Bitmap - Revision 2 (section 2.2.2.3.1.2.3) Secondary Drawing Order).

The color table entry MUST be in the range 0 to 5 (inclusive), while the bitmap cache ID MUST be in the range negotiated by the Revision 1 or 2 Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5). However, if the bitmap cache ID is set to TS_BITMAPCACHE_SCREEN_ID (0xFF), the cacheIndex field MUST contain the index of an entry in the Offscreen Bitmap Cache that contains the source bitmap.

nLeftRect (variable): The left coordinate of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nWidth (variable): The width of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nHeight (variable): The height of the blit rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop (1 byte): The index of the ternary raster operation to perform (see section 2.2.2.3.1.1.1.7).

nXSrc (variable): The X coordinate of the source rectangle within the source bitmap specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nYSrc (variable): Y coordinate of the source rectangle within the source bitmap specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BackColor (3 bytes): Background color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): Foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BrushOrgX (1 byte): An 8-bit signed integer. The X coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushOrgY (1 byte): An 8-bit signed integer. The Y coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushStyle (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushStyle** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushHatch (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushHatch** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushExtra (7 bytes): The contents and format of this field are the same as the **BrushExtra** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

cacheIndex (2 bytes): A 16-bit unsigned integer. The index of the source bitmap within the bitmap cache specified by the **cacheId** field. This value MUST be in the range negotiated by the Revision 1 or 2 Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5).

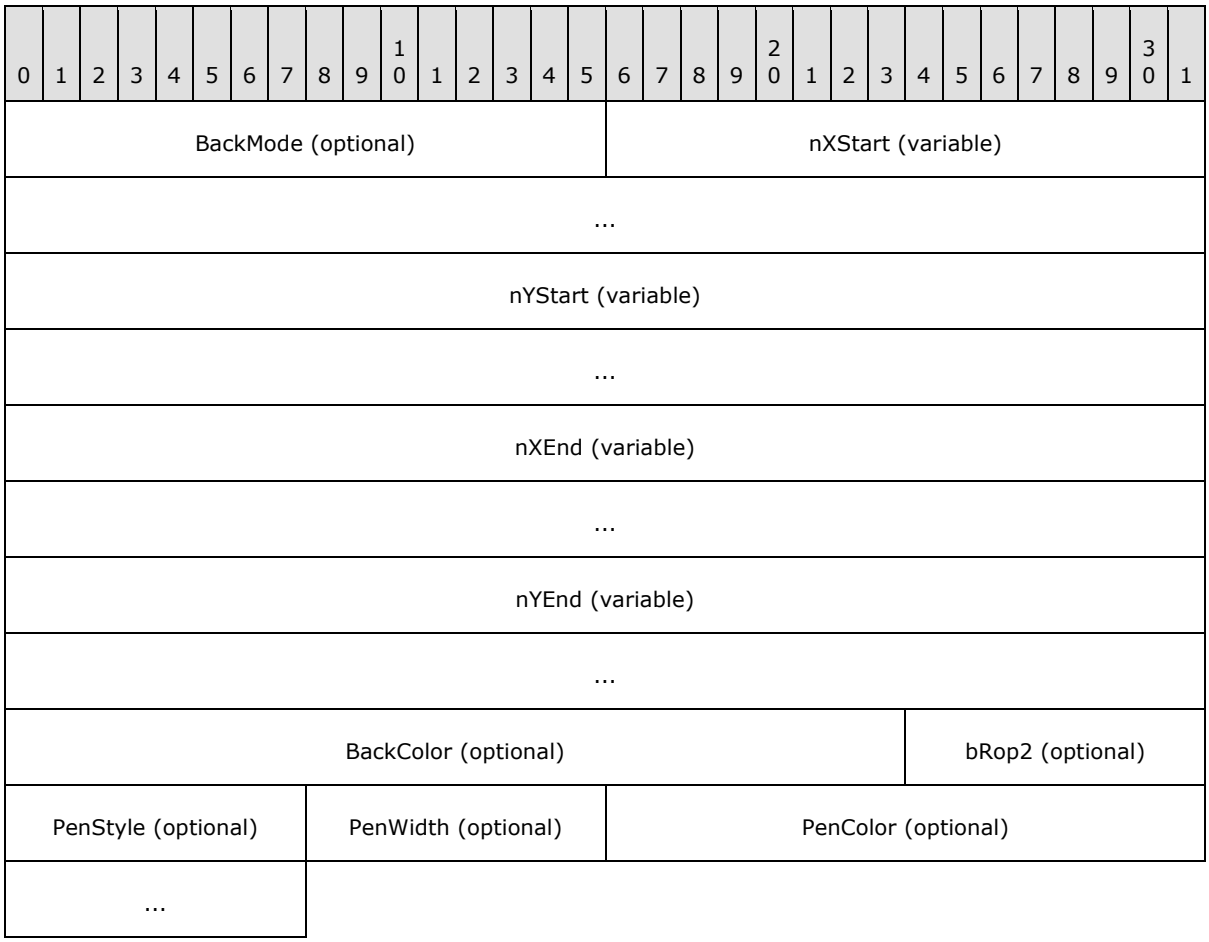
If this field is set to BITMAPCACHE_WAITING_LIST_INDEX (32767), the last bitmap cache entry MUST contain the bitmap.

If the bitmap cache ID (specified in the **cacheId** field) is set to TS_BITMAPCACHE_SCREEN_ID (0xFF), this field MUST contain the index of an entry in the Offscreen Bitmap Cache that contains the source bitmap.

2.2.2.3.1.1.2.11 LineTo (LINETO_ORDER)

The LineTo Primary Drawing Order encodes a single line drawing order that is restricted to solid color lines, one pixel wide.

```
Encoding order number: 9 (0x09)
Negotiation order number: 8 (0x08)
Number of fields: 10
Number of field encoding bytes: 2
Maximum encoded field length: 19 bytes
```



BackMode (2 bytes): A signed 16-bit integer. This field contains the background mix mode and MUST be one of the following values.

Value	Meaning
TRANSPARENT 0x0001	Background remains untouched.
OPAQUE 0x0002	Background is filled with the current background color before the pen is drawn.

nXStart (variable): The X coordinate of the starting point of the line specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nYStart (variable): The Y coordinate of the starting point of the line specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nXEnd (variable): The X coordinate of the ending point of the line specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nYEnd (variable): The Y coordinate of the ending point of the line specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BackColor (3 bytes): The background color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure. This field MUST be zeroed out.

bRop2 (1 byte): The binary raster operation to perform (see section 2.2.2.3.1.1.1.6).

PenStyle (1 byte): An 8-bit unsigned integer. The drawing style of the pen. This field MUST be set to PS_SOLID (0x00).

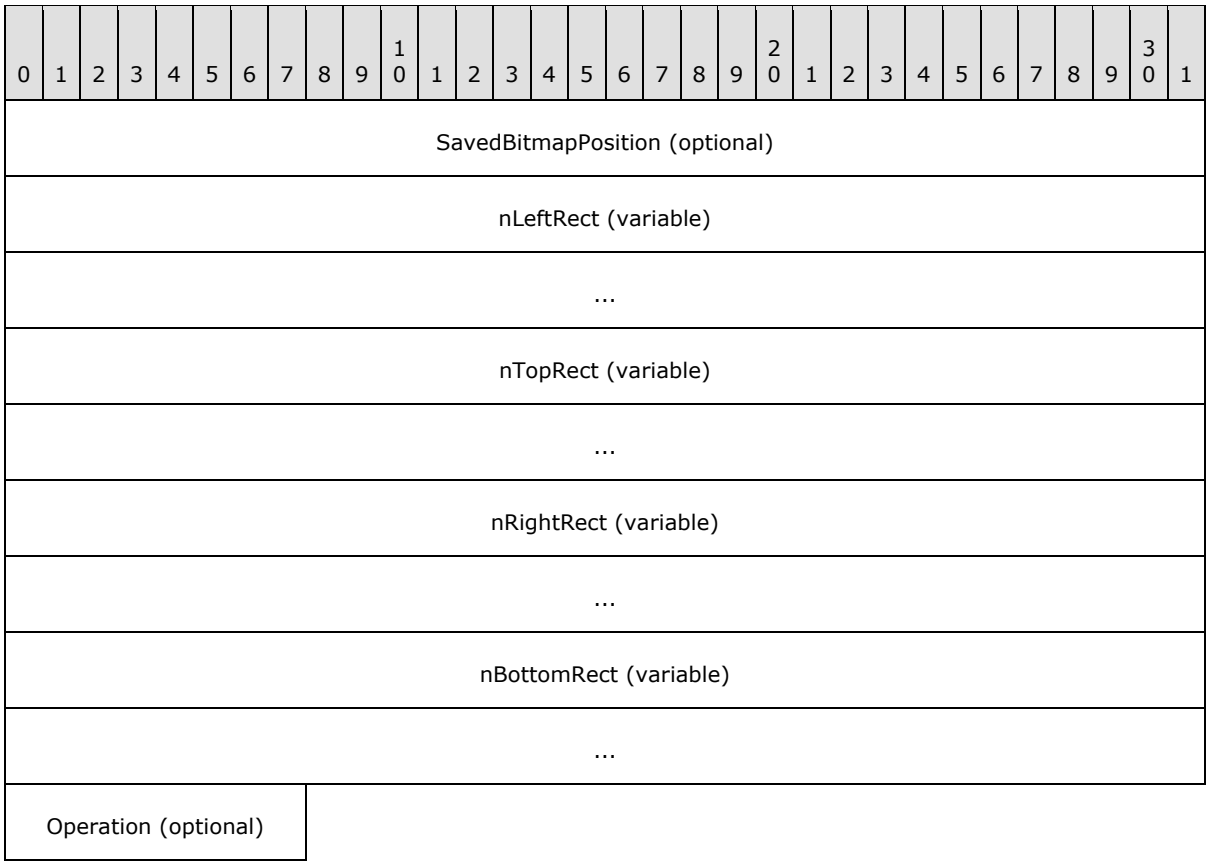
PenWidth (1 byte): An 8-bit unsigned integer. The width of the pen. This field MUST be set to 0x01.

PenColor (3 bytes): The foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

2.2.2.3.1.1.2.12 SaveBitmap (SAVEBITMAP_ORDER)

The SaveBitmap Primary Drawing Order encodes a rectangle of the screen image for saving or restoring by the client.

```
Encoding order number: 11 (0x0B)
Negotiation order number: 11 (0x0B)
Number of fields: 6
Number of field encoding bytes: 1
Maximum encoded field length: 13 bytes
```



SavedBitmapPosition (4 bytes): A 32-bit unsigned integer. Encoded start position of the rectangle in the Saved Bitmap that will be read from (in the case of a bitmap restore operation) or written to (in the case of a bitmap save operation), depending on the value of the **Operation** field.

The **SavedBitmapPosition** field is constructed by using the *desktopSaveXGranularity* and *desktopSaveYGranularity* values negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3). The size of the Save Bitmap MUST be 480 pixels wide by 480 pixels high.

Specification [T128] section 8.16.17 shows how to compute the value to insert into the **SavedBitmapPosition** field. When performing a save operation, the **SavedBitmapPosition** field contains the cumulative area of the virtual desktop rectangles already in the Save Bitmap (the area of the rectangle being written to the Save Bitmap is excluded). When performing a restore operation, the **SavedBitmapPosition** field contains the cumulative area of all the rectangles that were written to the Save Bitmap before the rectangle being restored was saved.

The functions used to compute the area using the negotiated X and Y granularity are:

```
AreaWidthInPixels = [(width + XGranularity - 1) / XGranularity] * XGranularity

AreaHeightInPixels = [(height + YGranularity - 1) / YGranularity] * YGranularity

Area = AreaWidthInPixels * AreaHeightInPixels
```

To determine the X and Y position in the Save Bitmap using the YGranularity and the Save Bitmap width of 480, the following functions are used:

```
Y = [SaveBitmapPosition / (480 * YGranularity)] * YGranularity

X = [SaveBitmapPosition - (Y * 480)] / YGranularity
```

An example of calculations to obtain the X and Y positions from the **SavedBitmapPosition** field are defined visually in section 4.7.

nLeftRect (variable): The left coordinate of the virtual desktop rectangle to save specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nTopRect (variable): The top coordinate of the virtual desktop rectangle to save specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nRightRect (variable): The right inclusive coordinate of the virtual desktop rectangle to save specified using a Coord Field (section 2.2.2.3.1.1.1.1).

nBottomRect (variable): The bottom inclusive coordinate of the virtual desktop rectangle to save specified using a Coord Field (section 2.2.2.3.1.1.1.1).

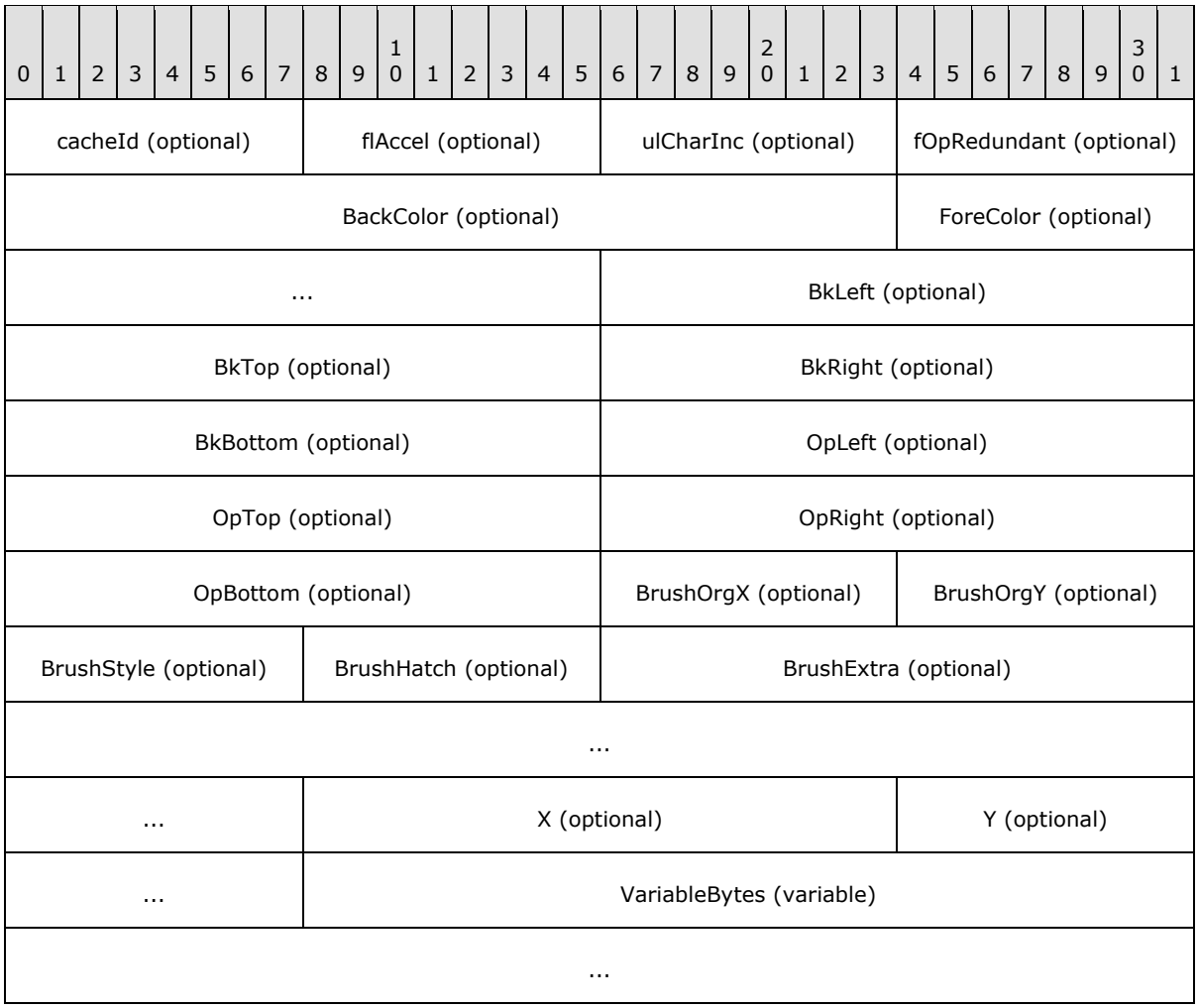
Operation (1 byte): An 8-bit unsigned integer. The operation to perform which MUST be one of the following values.

Value	Meaning
SV_SAVEBITS 0x00	Save bitmap operation.
SV_RESTOREBITS 0x01	Restore bitmap operation.

2.2.2.3.1.1.2.13 GlyphIndex (GLYPHINDEX_ORDER)

The GlyphIndex Primary Drawing Order encodes a set of glyph indices at a specified position.

```
Encoding order number: 27 (0x1B)
Negotiation order number: 27 (0x1B)
Number of fields: 22
Number of field encoding bytes: 3
Maximum encoded field length: 297 bytes
```



cacheId (1 byte): An 8-bit unsigned integer. The ID of the glyph cache in which the glyph is stored. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

flAccel (1 byte): An 8-bit unsigned integer. Accelerator flags. For glyph related terminology, see [WGFX] figures 14-17 and 15-1, and see [MSDN-FT]. MUST be one or more of the following values.

Value	Meaning
SO_FLAG_DEFAULT_PLACEMENT 0x01	This flag MUST be set for legacy reasons but is not used for any specific glyph processing.
SO_HORIZONTAL 0x02	Text is horizontal, left-to-right or right-to-left, depending on SO_REVERSED.
SO_VERTICAL 0x04	Text is vertical, top-to-bottom or bottom-to-top, depending on SO_REVERSED.
SO_REVERSED	Set if horizontal text is right-to-left or vertical text is bottom-

Value	Meaning
0x08	to-top.
SO_ZERO_BEARINGS 0x10	For a given glyph in the font, the A-width (left-side bearing) and C-width (right-side bearing) associated with the glyph have a value of zero.
SO_CHAR_INC_EQUAL_BM_BASE 0x20	For a given glyph in the font, the B-width associated with the glyph equals the advance width of the glyph.
SO_MAXEXT_EQUAL_BM_SIDE 0x40	The height of the bitmap associated with a given glyph in the font is always equal to the sum of the ascent and descent. This implies that the tops and bottoms of all glyph bitmaps lie on the same line in the direction of writing.

ulCharInc (1 byte): An 8-bit unsigned integer. Specifies whether or not the font is a fixed-pitch (monospace) font. If so, this member is equal to the advance width of the glyphs in pixels (see [WGFX] figures 14-17); if not, this field is set to 0x00. The minimum value for this field is 0x00 (inclusive), and the maximum value is 0xFF (inclusive).

fOpRedundant (1 byte): An 8-bit unsigned integer. A Boolean value indicating whether or not the opaque rectangle is redundant. Redundant, in this context, means that it is the same color as the background rectangle.

Value	Meaning
FALSE 0x00	Rectangle is not redundant.
TRUE 0x01	Rectangle is redundant.

BackColor (3 bytes): The text color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): Color of the opaque rectangle described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BkLeft (2 bytes): A 16-bit signed integer. The left coordinate of the text background rectangle.

BkTop (2 bytes): A 16-bit signed integer. The top coordinate of the text background rectangle.

BkRight (2 bytes): A 16-bit signed integer. The right coordinate of the text background rectangle.

BkBottom (2 bytes): A 16-bit signed integer. The bottom coordinate of the text background rectangle.

OpLeft (2 bytes): A 16-bit signed integer. The left coordinate of the opaque rectangle. This field MUST be set to 0 if the fOpRedundant flag is set.

OpTop (2 bytes): A 16-bit signed integer. The top coordinate of the opaque rectangle. This field MUST be set to 0 if the fOpRedundant flag is set.

OpRight (2 bytes): A 16-bit signed integer. The right coordinate of the opaque rectangle. This field MUST be set to 0 if the fOpRedundant flag is set.

OpBottom (2 bytes): A 16-bit signed integer. The bottom coordinate of the opaque rectangle. This field MUST be set to 0 if the fOpRedundant flag is set.

BrushOrgX (1 byte): An 8-bit signed integer. The X coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushOrgY (1 byte): An 8-bit signed integer. The Y coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushStyle (1 byte): An 8-bit unsigned integer. This field MUST be set to BS_SOLID (0x00), as the GlyphIndex Primary Drawing Order MUST only use solid color brushes to render the opaque rectangle.

BrushHatch (1 byte): An 8-bit unsigned integer. This field MUST be set to 0x00, as the GlyphIndex Primary Drawing Order MUST only use solid color brushes to render the opaque rectangle.

BrushExtra (7 bytes): This field is not used, as the GlyphIndex Primary Drawing Order MUST only use solid color brushes to render the opaque rectangle.

X (2 bytes): A 16-bit signed integer. The X coordinate of the point where the origin of the starting glyph should be positioned.

Y (2 bytes): A 16-bit signed integer. The Y coordinate of the point where the origin of the starting glyph should be positioned.

VariableBytes (variable): A One-Byte Header Variable Field (section 2.2.2.3.1.1.1.2) structure. This field can contain glyph fragments (which are composed of a series of one or more glyph cache indices) and instructions to use entries previously stored in the glyph fragment cache. Multiple glyph fragments can be contained in this field. The first byte of each fragment is either a USE (0xFE) operation byte or a glyph index (0x00 to 0x0FD) byte:

- A value of 0xFE (USE) indicates that a previously stored fragment MUST be displayed. The byte following the USE byte is the index in the fragment cache where the fragment is located. This fragment MUST be read and displayed. If the **ulCharInc** field is set to 0, the index byte MUST be followed by a delta byte that indicates the distance between two consecutive fragments measured in pixels from the beginning of the first fragment to the beginning of the next. If the distance is greater than 127 (0x7F), the value 0x80 is used, and the following two bytes contain the actual distance.

- If not preceded by 0xFE, a value of 0x00 to 0xFD identifies a glyph stored at the given index in the glyph cache. Multiple glyphs can be sent at one time. If the **ulCharInc** field is set to 0, the index byte is followed by a delta byte that indicates the distance between two consecutive glyphs, measured in pixels from the beginning of the first glyph to the beginning of the next. If the distance is greater than 127 (0x7F), the value 0x80 is used, and the following two bytes contain the actual distance.

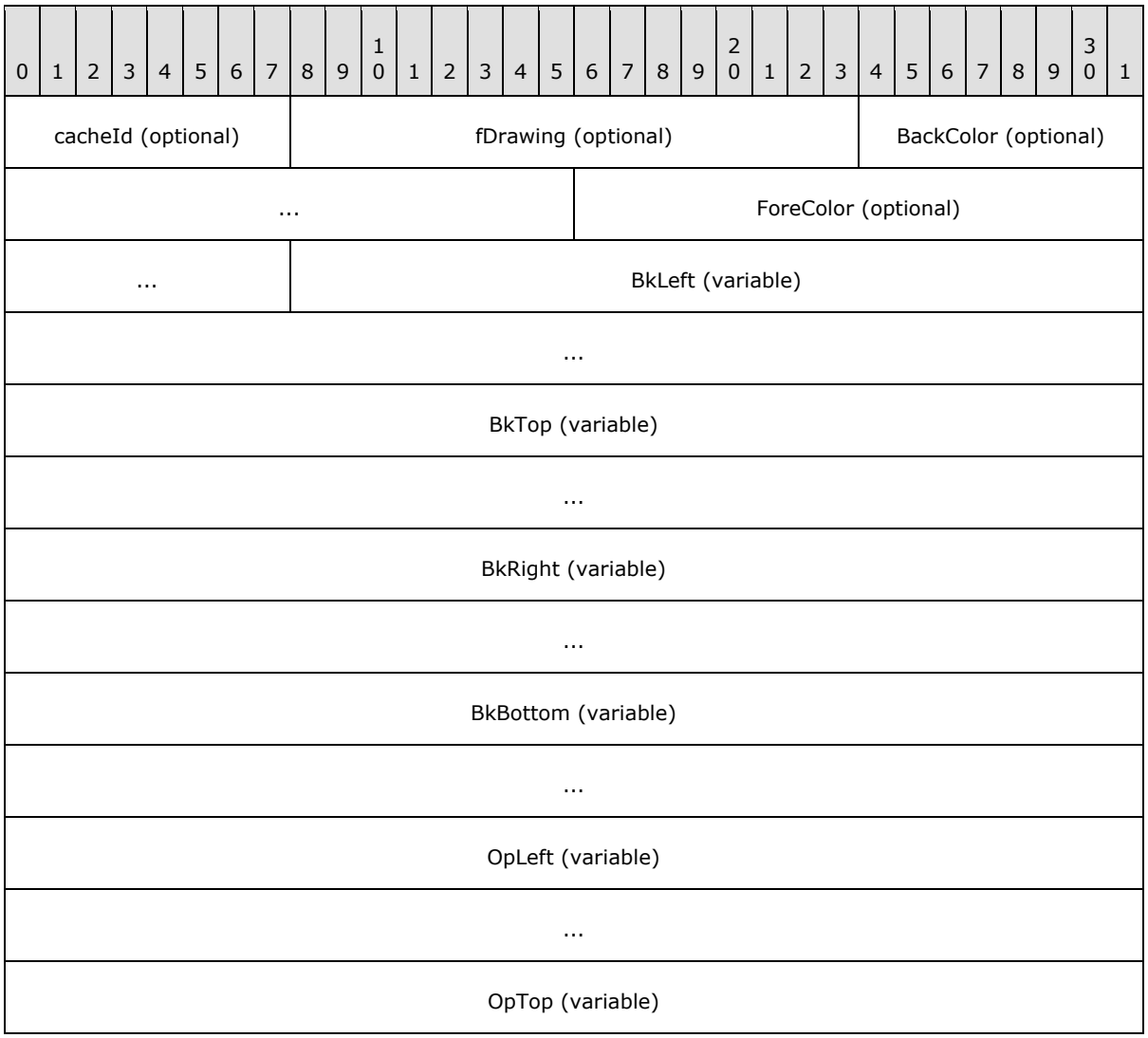
If a series of glyph indices ends with an ADD (0xFF) operation byte, the preceding glyph information MUST be collected, displayed, and then stored in the fragment cache. The byte following the ADD byte is the index of the cache where the fragment MUST be stored. A final byte that indicates the size of the fragment follows the index byte (the ADD byte, index byte, and size byte MUST NOT be counted when calculating the value of the size byte).

All glyph and fragment cache indices MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

2.2.2.3.1.1.2.14 FastIndex (FASTINDEX_ORDER)

The FastIndex Primary Drawing Order encodes a set of glyph indices at a specified position. This order is an improved version of the GlyphIndex (section 2.2.2.3.1.1.2.13) Primary Drawing Order. The regular GlyphIndex order contains five brush fields that the FastIndex order does not: **BrushOrgX**, **BrushOrgY**, **BrushStyle**, **BrushHatch**, and **BrushExtra**. These extra fields MUST all be implicitly assumed to exist, and contain default values of zero (implying that a solid color brush will be used).

```
Encoding order number: 19 (0x13)
Negotiation order number: 19 (0x13)
Number of fields: 15
Number of field encoding bytes: 2
Maximum encoded field length: 285 bytes
```



...	
OpRight (variable)	
...	
OpBottom (variable)	
...	
X (optional)	Y (optional)
VariableBytes (variable)	
...	

cacheId (1 byte): An 8-bit unsigned integer. The ID of the glyph cache in which the glyph is stored. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

fDrawing (2 bytes): A 16-bit unsigned integer. Combined **flAccel** and **ulCharInc** fields from the GlyphIndex (section 2.2.2.3.1.1.2.13) Primary Drawing Order. The high-order byte contains the **flAccel** field, and the low-order byte contains the **ulCharInc** field.

BackColor (3 bytes): The text color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): Opaque rectangle color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BkLeft (variable): Left coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BkTop (variable): Top coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BkRight (variable): Right coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BkBottom (variable): Bottom coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

OpLeft (variable): Left coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to 0 if it is the same as **BkLeft**.

OpTop (variable): The top coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

This field MUST contain opaque rectangle encoding flags if the **OpBottom** field is set to -32768. In this case, **OpTop** holds the encoding flags in the low four bits.

Value	Meaning
OPRECT_BOTTOM_ABSENT 0x01	Bottom coordinate of the opaque rectangle is the same as **BkBottom**.
OPRECT_RIGHT_ABSENT 0x02	Right coordinate of the opaque rectangle is the same as **BkRight**.
OPRECT_TOP_ABSENT 0x04	Top coordinate of the opaque rectangle is the same as **BkTop**.
OPRECT_LEFT_ABSENT 0x08	Left coordinate of the opaque rectangle is the same as **BkLeft**.

The only valid combinations of the encoding flags that are currently supported are:

- 0x0F: The left, top, right, and bottom coordinates of the opaque rectangle all match the background text rectangle and are absent.
- 0x0D: The left, top, and bottom coordinates of the opaque rectangle all match the background text rectangle and are absent. The actual value of the right coordinate is present.

OpRight (variable): The right coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to 0 if it is the same as **BkRight**.

OpBottom (variable): The bottom coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to -32768 if **OpTop** contains opaque rectangle encoding flags (see the **OpTop** field).

X (2 bytes): A 16-bit signed integer. The X coordinate of the point where the origin of the starting glyph should be positioned specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to -32768 if it is the same as **BkLeft**.

Y (2 bytes): A 16-bit signed integer. The Y coordinate of the point where the origin of the starting glyph should be positioned specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to -32768 if it is the same as **BkTop**.

VariableBytes (variable): A One-Byte Header Variable Field (section 2.2.2.3.1.1.1.2) structure. The contents and format of this field are the same as the **VariableBytes** field of the GlyphIndex (section 2.2.2.3.1.1.2.13) Primary Drawing Order.

2.2.2.3.1.1.2.15 FastGlyph (FASTGLYPH_ORDER)

The FastGlyph Primary Drawing Order encodes a single glyph at a specified position. This primary drawing order is a fast way of outputting a single glyph, and bypasses having to send a Cache Glyph Secondary Drawing Order (see sections 2.2.2.3.1.2.5 and 2.2.2.3.1.2.6) followed by a GlyphIndex (section 2.2.2.3.1.1.2.13) or FastIndex (section 2.2.2.3.1.1.2.14) Primary Drawing Order.

Encoding order number: 24 (0x18)

```
Negotiation order number: 24 (0x18)
Number of fields: 15
Number of field encoding bytes: 2
Maximum encoded field length: 285 bytes
```

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cacheId (optional)								fDrawing (optional)																BackColor (optional)							
...																				ForeColor (optional)											
...								BkLeft (optional)																BkTop (optional)							
...								BkRight (optional)																BkBottom (optional)							
...								OpLeft (optional)																OpTop (optional)							
...								OpRight (optional)																OpBottom (optional)							
...								X (optional)																Y (optional)							
...								VariableBytes (variable)																									
...																															

cacheId (1 byte): An 8-bit unsigned integer. The ID of the glyph cache in which the glyph is stored. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

fDrawing (2 bytes): A 16-bit unsigned integer. Combined **flAccel** and **ulCharInc** fields from the GlyphIndex (section 2.2.2.3.1.1.2.13) Primary Drawing Order. The high-order byte contains the **flAccel** field, and the low-order byte contains the **ulCharInc** field.

BackColor (3 bytes): Text color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): The opaque rectangle color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BkLeft (2 bytes): The left coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BkTop (2 bytes): The top coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BkRight (2 bytes): The right coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BkBottom (2 bytes): The bottom coordinate of the text background rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1).

OpLeft (2 bytes): The left coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to 0 if it is the same as **BkLeft**.

OpTop (2 bytes): The top coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST contain opaque rectangle encoding flags if the **OpBottom** field is set to -32768. For details on the opaque rectangle encoding flags, see the **OpTop** field of the FastIndex (section 2.2.2.3.1.1.2.14) Primary Drawing Order.

OpRight (2 bytes): The right coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to 0 if it is the same as **BkRight**.

OpBottom (2 bytes): The bottom coordinate of the opaque rectangle specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to -32768 if **OpTop** contains opaque rectangle encoding flags (see the **OpTop** field).

X (2 bytes): A 16-bit signed integer. The X coordinate of the point where the origin of the starting glyph should be positioned, specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to -32768 if it is the same as **BkLeft**.

Y (2 bytes): A 16-bit signed integer. The Y coordinate of the point where the origin of the starting glyph should be positioned, specified using a Coord Field (section 2.2.2.3.1.1.1.1). This field MUST be set to -32768 if it is the same as **BkTop**.

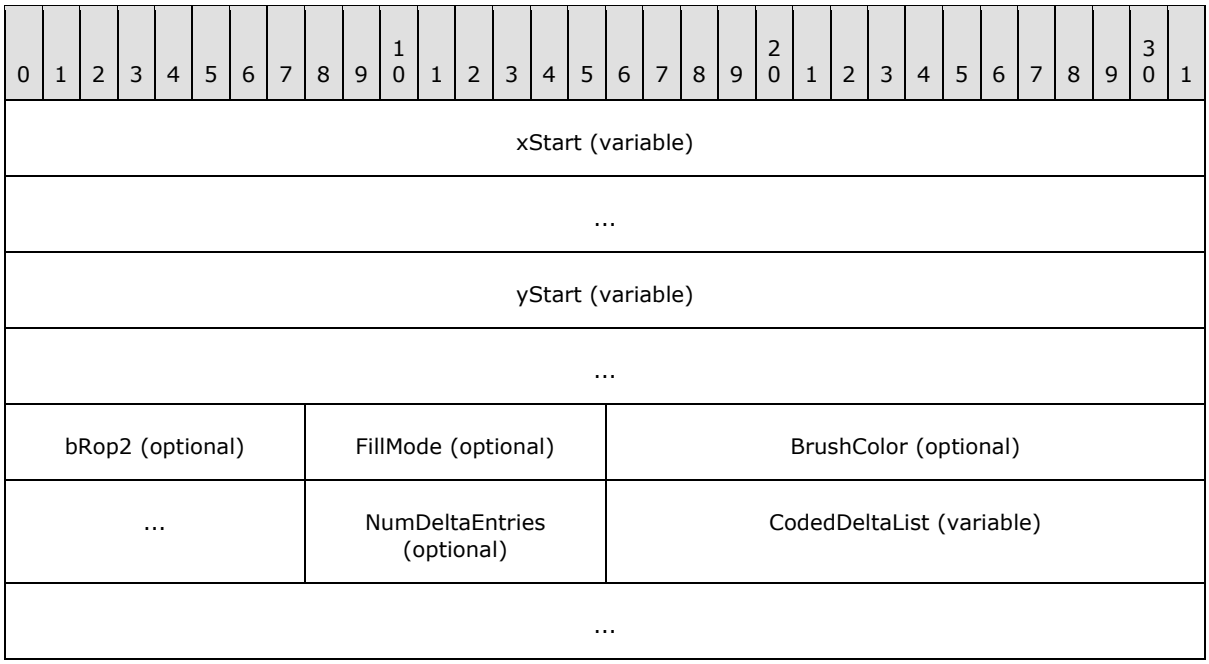
VariableBytes (variable): A One-Byte Header Variable Field (section 2.2.2.3.1.1.1.2) structure. If the size of this variable-bytes field is 1 byte, it contains a 1-byte glyph cache index from which the glyph data MUST be retrieved. However, if the size is larger than 1 byte, this field contains a Cache Glyph Data - Revision 2 (section 2.2.2.3.1.2.6.1) structure. The glyph data MUST be stored in the glyph cache specified by the **cacheId** field in the entry indicated by the **cacheIndex** field of the glyph data. The Cache Glyph Data - Revision 2 structure MUST be followed by two bytes of padding that MAY<2> optionally contain a little-endian ordered word representing the Unicode value of the glyph.

All glyph cache indices MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

2.2.2.3.1.1.2.16 PolygonSC (POLYGON_SC_ORDER)

The PolygonSC Primary Drawing Order encodes a solid-color polygon consisting of two or more vertices connected by straight lines.

```
Encoding order number: 20 (0x14)
Negotiation order number: 20 (0x14)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 249 bytes
```



xStart (variable): The X coordinate of the starting point of the polygon path specified using a Coord Field (section 2.2.2.3.1.1.1.1).

yStart (variable): The Y coordinate of the starting point of the polygon path specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop2 (1 byte): The binary raster operation to perform (see section 2.2.2.3.1.1.1.6).

FillMode (1 byte): The polygon filling algorithm described using a Fill Mode (section 2.2.2.3.1.1.1.9) structure.

BrushColor (3 bytes): Foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

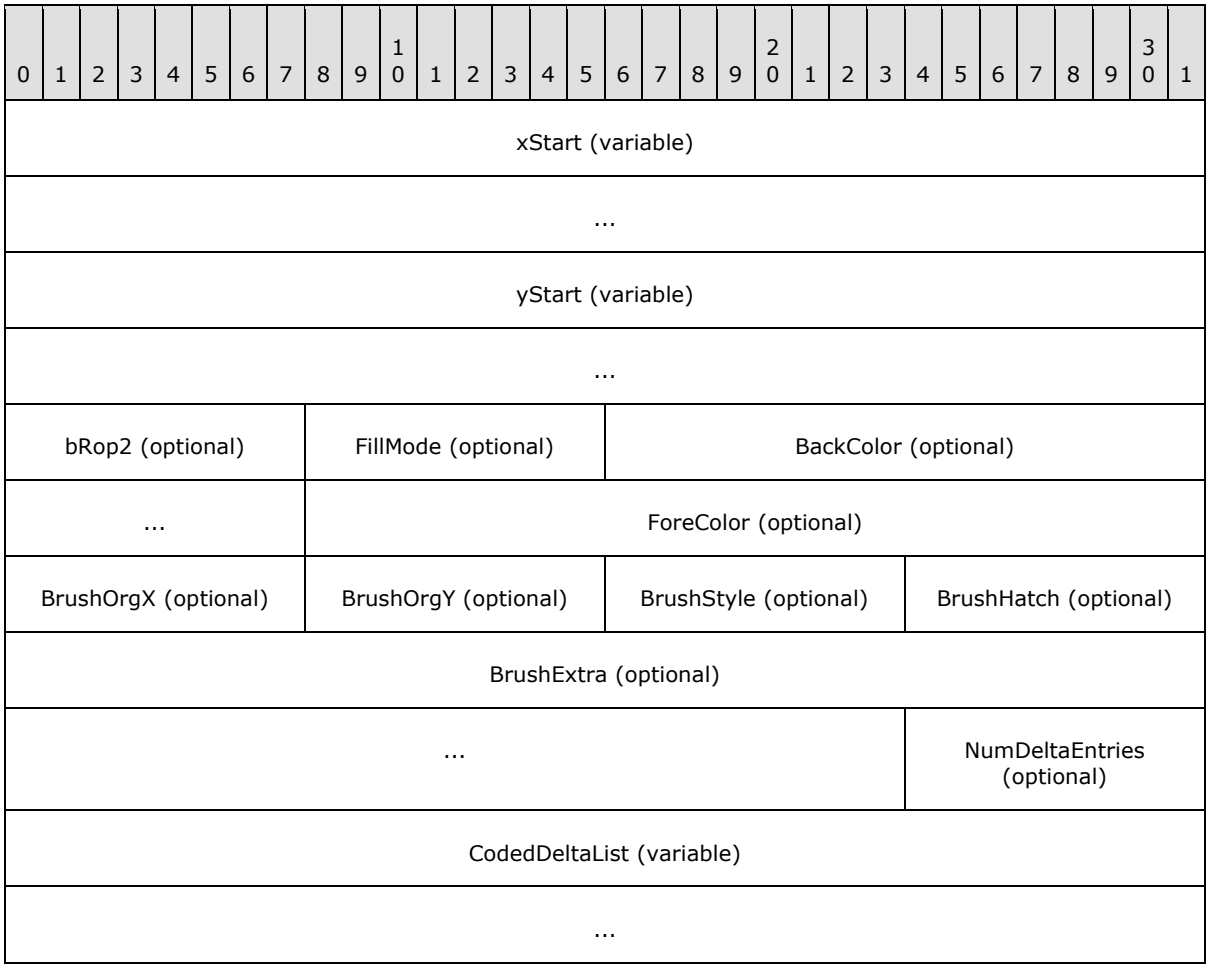
NumDeltaEntries (1 byte): An 8-bit unsigned integer. The number of points along the polygon path described by the **CodedDeltaList** field.

CodedDeltaList (variable): A One-Byte Header Variable Field (section 2.2.2.3.1.1.1.2) structure that encapsulates a Delta-Encoded Points (section 2.2.2.3.1.1.1.4) structure that contains the points along the polygon path. The number of points described by the Delta-Encoded Points structure is specified by the **NumDeltaEntries** field.

2.2.2.3.1.1.2.17 PolygonCB (POLYGON_CB_ORDER)

The PolygonCB Primary Drawing Order encodes a color brush polygon consisting of two or more vertices connected by straight lines.

```
Encoding order number: 21 (0x15)
Negotiation order number: 21 (0x15)
Number of fields: 13
Number of field encoding bytes: 2
Maximum encoded field length: 263 bytes
```



xStart (variable): The X coordinate of the starting point of the polygon path specified using a Coord Field (section 2.2.2.3.1.1.1.1).

yStart (variable): The Y coordinate of the starting point of the polygon path specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop2 (1 byte): The binary raster operation to perform (see section 2.2.2.3.1.1.1.6). The ROP2 field for the PolygonCB Primary Drawing Order has two bitfields within it. The low 5 bits (mask 0x1F) identify the real ROP2 operation. The high bit (mask 0x80) indicates whether the background drawing mode should be set to TRANSPARENT or OPAQUE (see section 2.2.2.3.1.1.2.11). The background drawing mode is only significant if the **BrushStyle** field indicates that the brush is a BS_HATCHED (0x02) or BS_PATTERN (0x03) brush.

FillMode (1 byte): The polygon filling algorithm described using a Fill Mode (section 2.2.2.3.1.1.1.9) structure.

BackColor (3 bytes): The background color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): The foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BrushOrgX (1 byte): An 8-bit signed integer. The X coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushOrgY (1 byte): An 8-bit signed integer. The Y coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushStyle (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushStyle** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushHatch (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushHatch** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushExtra (7 bytes): A 7-byte, byte array. The contents and format of this field are the same as the **BrushExtra** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

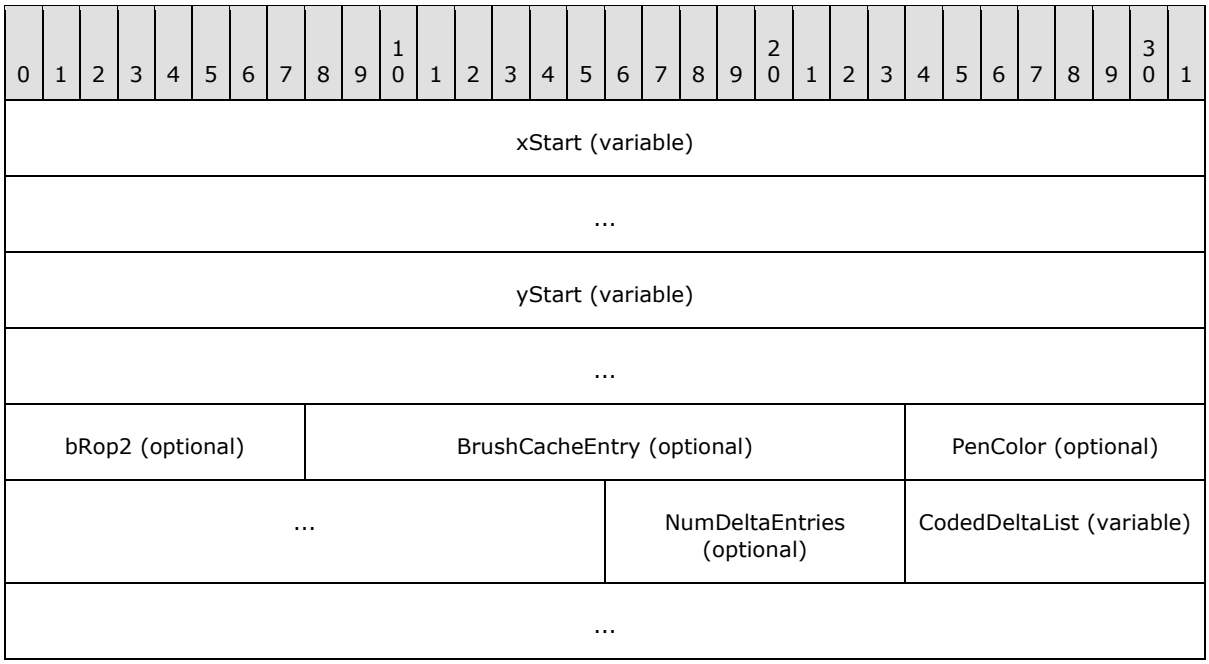
NumDeltaEntries (1 byte): An 8-bit unsigned integer. The number of points along the polygon path described by the **CodedDeltaList** field.

CodedDeltaList (variable): A One-Byte Header Variable Field (section 2.2.2.3.1.1.1.2) structure that encapsulates a Delta-Encoded Points (section 2.2.2.3.1.1.1.4) structure that contains the points along the polygon path. The number of points described by the Delta-Encoded Points structure is specified by the **NumDeltaEntries** field.

2.2.2.3.1.1.2.18 Polyline (POLYLINE_ORDER)

The Polyline Primary Drawing Order encodes a solid color polyline consisting of two or more vertices connected by straight lines.

```
Encoding order number: 22 (0x16)
Negotiation order number: 22 (0x16)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 148 bytes
```



xStart (variable): The X coordinate of the starting point of the polygon path specified using a Coord Field (section 2.2.2.3.1.1.1.1).

yStart (variable): The Y coordinate of the starting point of the polygon path specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop2 (1 byte): The binary raster operation to perform (see section 2.2.2.3.1.1.1.6).

BrushCacheEntry (2 bytes): A 16-bit unsigned integer. The brush cache entry. This field is unused (as only solid color polylines are drawn) and SHOULD<3> be set to 0x0000.

PenColor (3 bytes): The foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

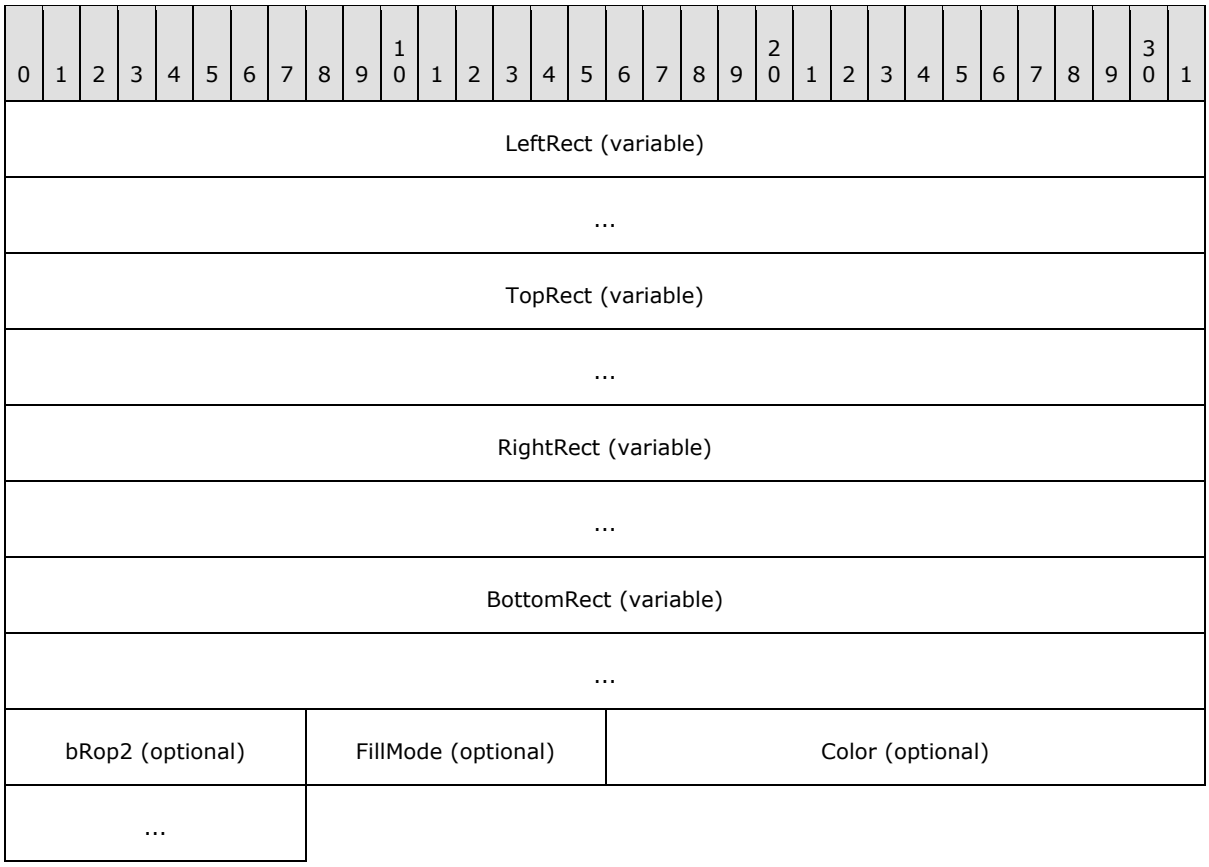
NumDeltaEntries (1 byte): An 8-bit unsigned integer. The number of points along the polyline path described by the **CodedDeltaList** field.

CodedDeltaList (variable): A One-Byte Header Variable Field (section 2.2.2.3.1.1.1.2) structure that encapsulates a Delta-Encoded Points (section 2.2.2.3.1.1.1.4) structure that contains the points along the polyline path. The number of points described by the Delta-Encoded Points structure is specified by the **NumDeltaEntries** field.

2.2.2.3.1.1.2.19 EllipseSC (ELLIPSE_SC_ORDER)

The EllipseSC Primary Drawing Order encodes a single solid color ellipse.

```
Encoding order number: 25 (0x19)
Negotiation order number: 25 (0x19)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 13 bytes
```



LeftRect (variable): The left coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

TopRect (variable): The top coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

RightRect (variable): The right coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BottomRect (variable): The bottom coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop2 (1 byte): The binary raster operation to perform (see section 2.2.2.3.1.1.1.6).

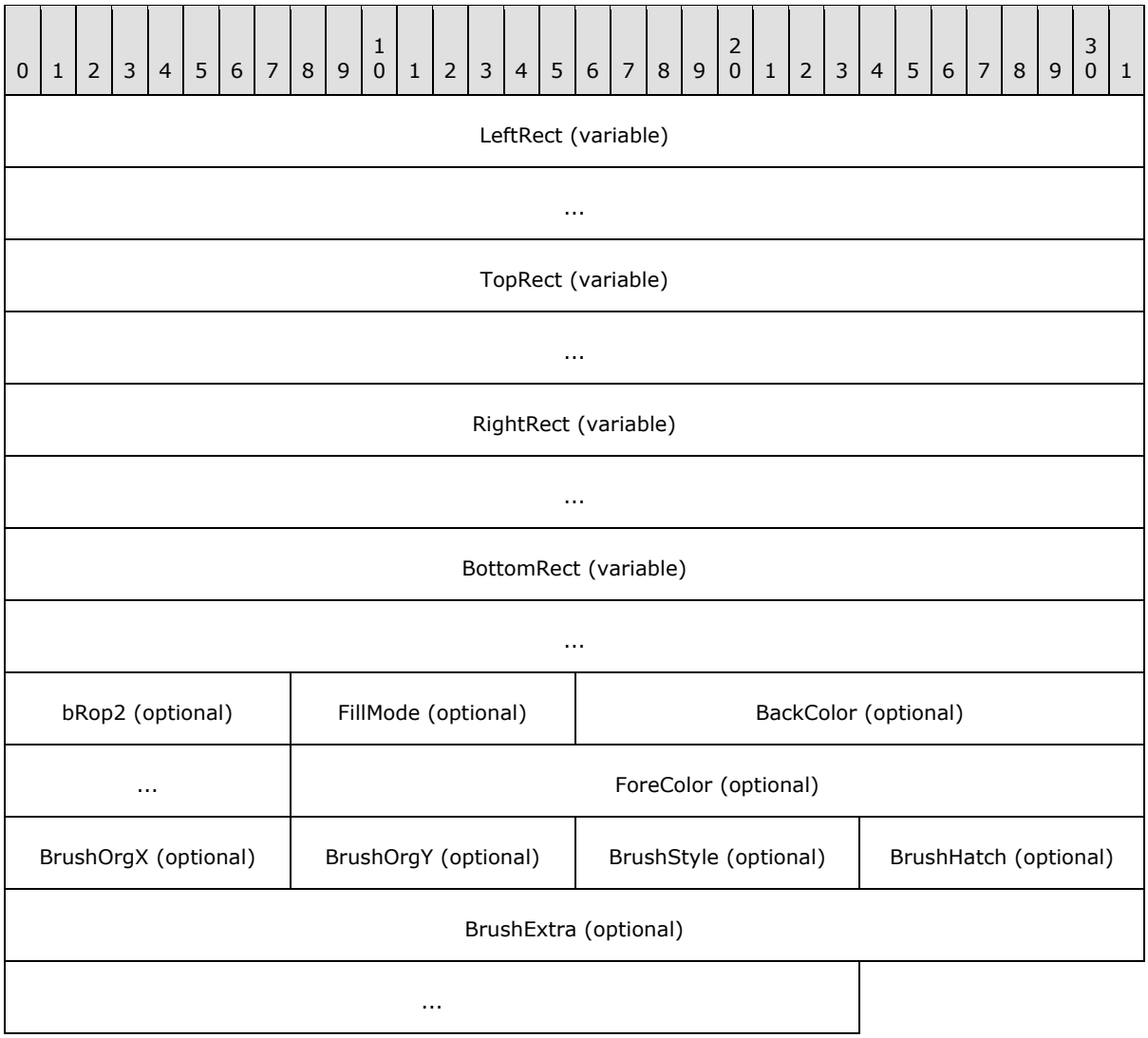
FillMode (1 byte): The ellipse filling algorithm described using a Fill Mode (section 2.2.2.3.1.1.1.9) structure.

Color (3 bytes): The foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

2.2.2.3.1.1.2.20 EllipseCB (ELLIPSE_CB_ORDER)

The EllipseCB Primary Drawing Order encodes a color brush ellipse.

```
Encoding order number: 26 (0x1A)
Negotiation order number: 26 (0x1A)
Number of fields: 13
Number of field encoding bytes: 2
Maximum encoded field length: 27 bytes
```



LeftRect (variable): The left coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

TopRect (variable): The top coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

RightRect (variable): The right coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

BottomRect (variable): The bottom coordinate of the inclusive rectangle for the ellipse specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bRop2 (1 byte): The binary raster operation to perform (see section 2.2.2.3.1.1.1.6). The ROP2 field for the EllipseCB Primary Drawing Order has two bitfields within it. The low 5 bits (mask 0x1F) identify the real ROP2 operation. The high bit (mask 0x80) indicates whether the background drawing mode should be set to TRANSPARENT or OPAQUE (see section 2.2.2.3.1.1.2.11). The background drawing mode is only significant if the **BrushStyle** field indicates that the brush is a BS_HATCHED (0x02) or BS_PATTERN (0x03) brush.

FillMode (1 byte): The ellipse filling algorithm described using a Fill Mode (section 2.2.2.3.1.1.1.9) structure.

BackColor (3 bytes): The background color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

ForeColor (3 bytes): The foreground color described using a Generic Color (section 2.2.2.3.1.1.1.8) structure.

BrushOrgX (1 byte): An 8-bit signed integer. The X coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushOrgY (1 byte): An 8-bit signed integer. The Y coordinate of the point where the top leftmost pixel of a brush pattern MUST be anchored.

BrushStyle (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushStyle** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

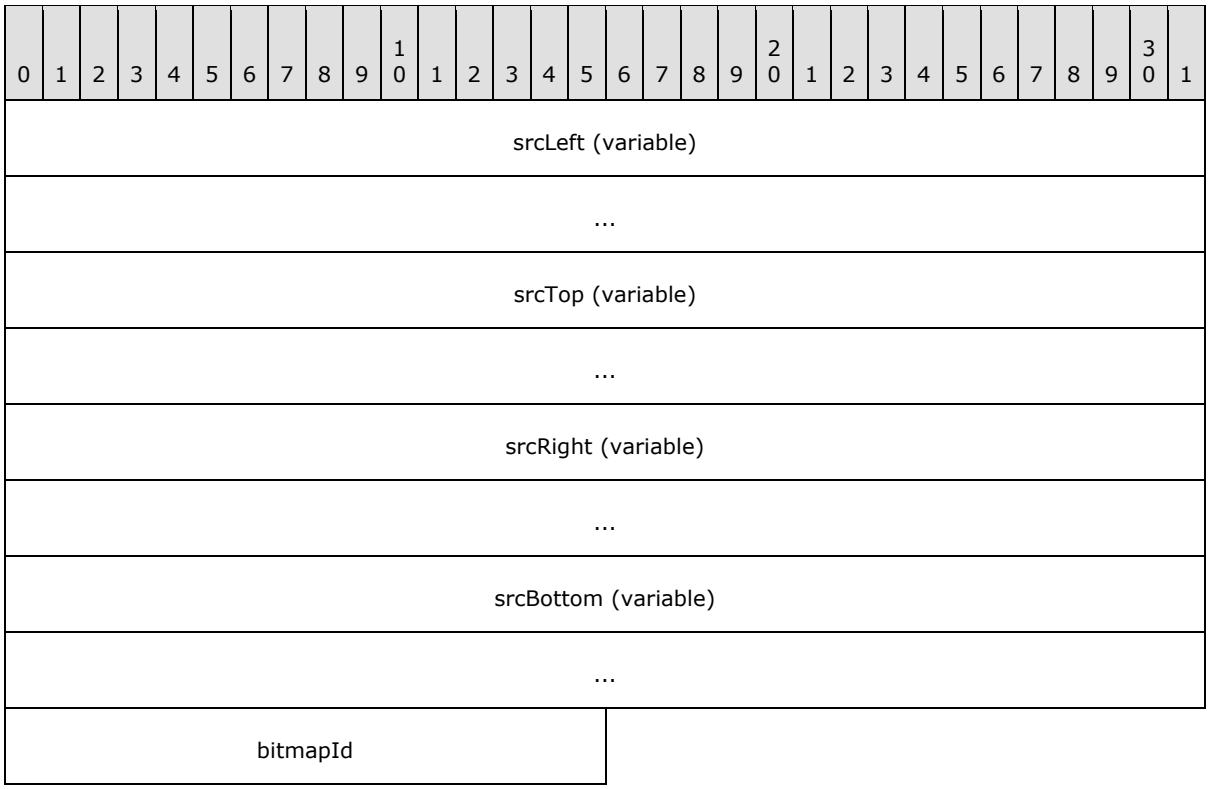
BrushHatch (1 byte): An 8-bit unsigned integer. The contents and format of this field are the same as the **BrushHatch** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

BrushExtra (7 bytes): A 7-byte, byte array. The contents and format of this field are the same as the **BrushExtra** field of the PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

2.2.2.3.1.1.2.21 DrawNineGrid (DRAWNINEGRID_ORDER)

The DrawNineGrid Primary Drawing Order encodes a single NineGrid drawing command with a single bounding rectangle.

```
Encoding order number: 7 (0x07)
Negotiation order number: 7 (0x07)
Number of fields: 5
Number of field encoding bytes: 1
Maximum encoded field length: 10 bytes
```



srcLeft (variable): The left coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

srcTop (variable): The top coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

srcRight (variable): The right coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

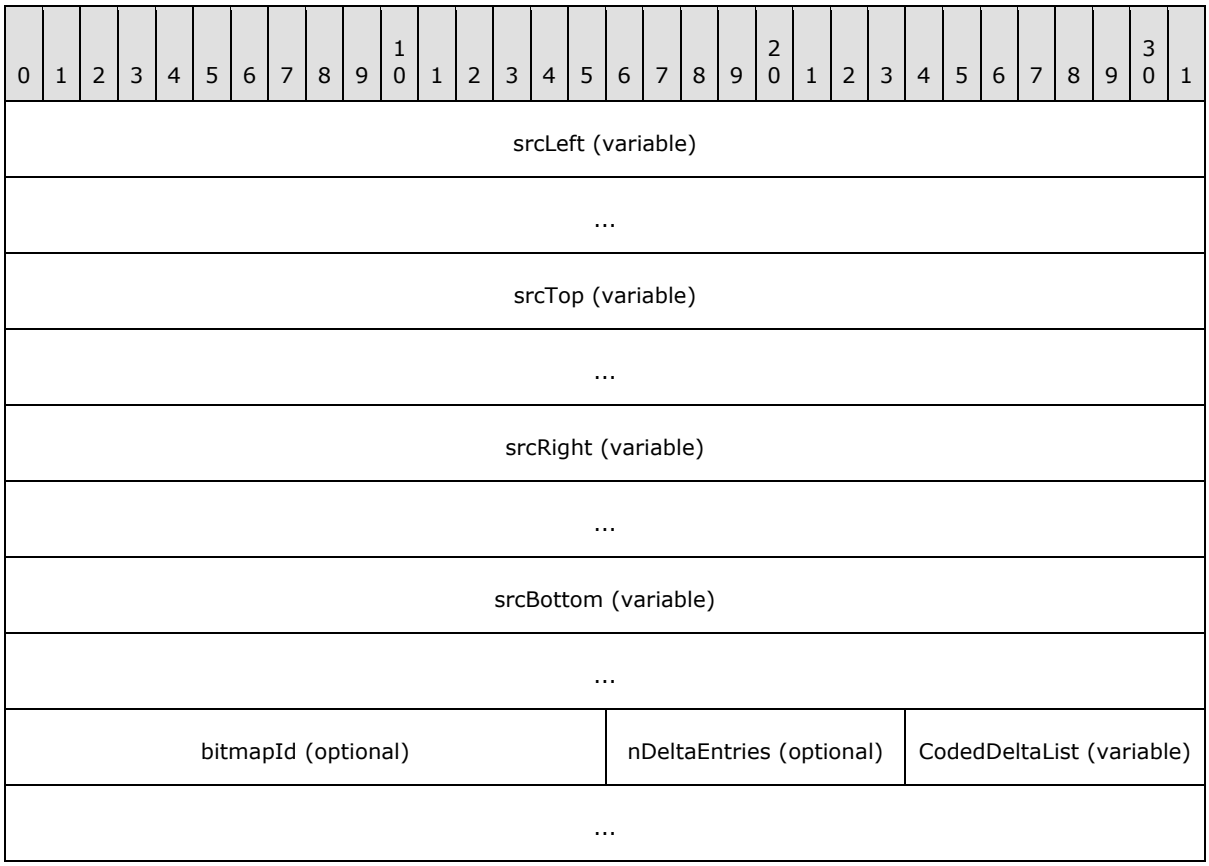
srcBottom (variable): The bottom coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bitmapId (2 bytes): A 16-bit unsigned integer. The NineGrid Bitmap Cache entry ID where the bitmap and the associated NineGrid transformation information reside. This ID MUST be in the range negotiated by the DrawNineGrid Cache Capability Set (section 2.2.1.2). The bitmap and transformation information MUST have already been cached in response to a Create NineGrid Bitmap (section 2.2.2.3.1.3.4) Alternate Secondary Drawing Order.

2.2.2.3.1.1.2.22 MultiDrawNineGrid (MULTI_DRAWNINEGRID_ORDER)

The MultiDrawNineGrid Primary Drawing Order encodes a single NineGrid drawing command with multiple clipping rectangles.

```
Encoding order number: 8 (0x08)
Negotiation order number: 9 (0x09)
Number of fields: 7
Number of field encoding bytes: 1
Maximum encoded field length: 396 bytes
```



srcLeft (variable): The left coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

srcTop (variable): The top coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

srcRight (variable): The right coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

srcBottom (variable): The bottom coordinate of the clipping rectangle to be applied to the bitmap stored at the entry given by the **bitmapId** field. The coordinate is specified using a Coord Field (section 2.2.2.3.1.1.1.1).

bitmapId (2 bytes): A 16-bit unsigned integer. The NineGrid Bitmap Cache entry ID where the bitmap and the associated NineGrid transformation information reside. This ID MUST be in the range negotiated by the DrawNineGrid Cache Capability Set (section 2.2.1.2). The bitmap and transformation information MUST have already been cached in response to a Create NineGrid Bitmap (section 2.2.2.3.1.3.4) Alternate Secondary Drawing Order.

nDeltaEntries (1 byte): An 8-bit unsigned integer. The number of bounding rectangles described by the **CodedDeltaList** field.

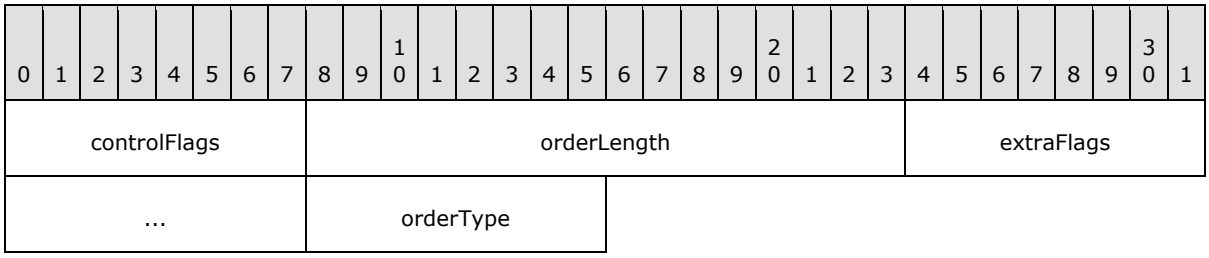
CodedDeltaList (variable): A Two-Byte Header Variable Field (section 2.2.2.3.1.1.1.3) structure that encapsulates a Delta-Encoded Rectangles (section 2.2.2.3.1.1.1.5) structure that contains bounding rectangles to use when rendering the order. The number of rectangles described by the Delta-Encoded Rectangles structure is specified by the **nDeltaEntries** field.

2.2.2.3.1.2 Secondary Drawing Orders

2.2.2.3.1.2.1 Common Data Types

2.2.2.3.1.2.1.1 Secondary Drawing Order Header (SECONDARY_DRAWING_ORDER_HEADER)

The SECONDARY_DRAWING_ORDER_HEADER structure is included in all secondary drawing orders.



controlFlags (1 byte): An 8-bit unsigned integer. The control byte that identifies the class of the drawing order. This field MUST contain the TS_STANDARD (0x01) and TS_SECONDARY (0x02) flags to indicate that the order is a secondary drawing order (see section 2.2.2.2.1).

orderLength (2 bytes): A 16-bit unsigned integer. The length in bytes of the secondary drawing order, including the size of the header. When encoding the order, the value in the **orderLength** field MUST be 13 bytes less than the actual value. Hence, when decoding the order, the **orderLength** field MUST be adjusted by adding 13 bytes. These adjustments are for historical reasons.

extraFlags (2 bytes): A 16-bit unsigned integer. Flags specific to each secondary drawing order.

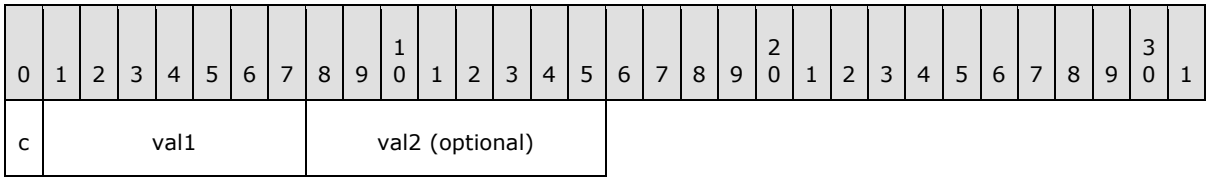
orderType (1 byte): An 8-bit unsigned integer. Identifies the type of secondary drawing order. MUST be one of the following values.

Value	Meaning
TS_CACHE_BITMAP_UNCOMPRESSED 0x00	Cache Bitmap - Revision 1 (section 2.2.2.3.1.2.2) Secondary Drawing Order with an uncompressed bitmap.
TS_CACHE_COLOR_TABLE	Cache Color Table (section 2.2.2.3.1.2.4) Secondary

Value	Meaning
0x01	Drawing Order.
TS_CACHE_BITMAP_COMPRESSED 0x02	Cache Bitmap - Revision 1 (section 2.2.2.3.1.2.2) Secondary Drawing Order with a compressed bitmap.
TS_CACHE_GLYPH 0x03	Cache Glyph - Revision 1 (section 2.2.2.3.1.2.5) or Cache Glyph - Revision 2 (section 2.2.2.3.1.2.6) Secondary Drawing Order. The version is indicated by the extraFlags field.
TS_CACHE_BITMAP_UNCOMPRESSED_REV2 0x04	Cache Bitmap - Revision 2 (section 2.2.2.3.1.2.3) Secondary Drawing Order with an uncompressed bitmap.
TS_CACHE_BITMAP_COMPRESSED_REV2 0x05	Cache Bitmap - Revision 2 (section 2.2.2.3.1.2.3) Secondary Drawing Order with a compressed bitmap.
TS_CACHE_BRUSH 0x07	Cache Brush (section 2.2.2.3.1.2.7) Secondary Drawing Order.

2.2.2.3.1.2.1.2 Two-Byte Unsigned Encoding (TWO_BYTE_UNSIGNED_ENCODING)

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



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

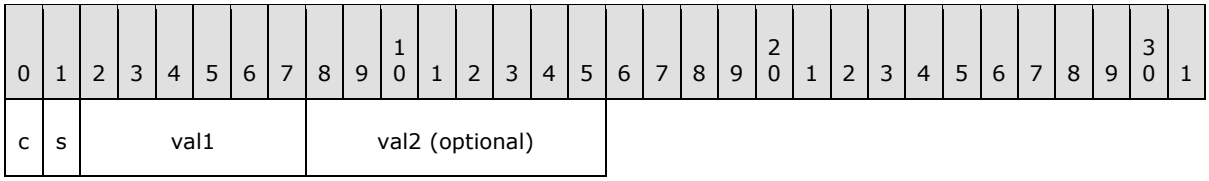
Value	Meaning
ONE_BYTE_VAL 0	Implies that the optional **val2** field is not present. Hence, the structure is 1 byte in size.
TWO_BYTE_VAL 1	Implies that the optional **val2** field is present. Hence, the structure is 2 bytes in size.

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

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

2.2.2.3.1.2.1.3 Two-Byte Signed Encoding (TWO_BYTE_SIGNED_ENCODING)

The TWO_BYTE_SIGNED_ENCODING structure is used to encode a value in the range -0x3FFF to 0x3FFF, using a variable number of bytes. For example, -0x1A1B is encoded as { 0xDA, 0x1B }, and -0x0002 is encoded as { 0x42 }. The most significant bits of the first byte encode the number of bytes in the structure and the sign.



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

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

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

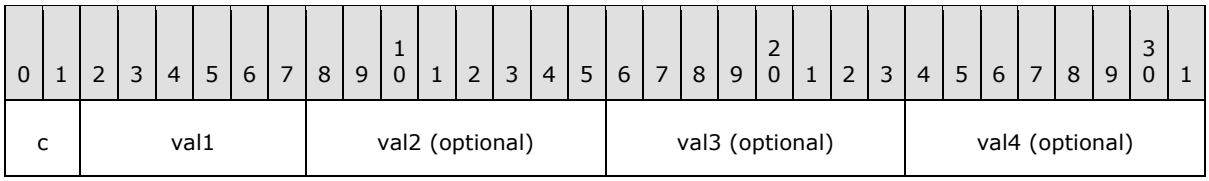
Value	Meaning
POSITIVE_VAL 0	Implies that the value represented by this structure is positive.
NEGATIVE_VAL 1	Implies that the value represented by this structure is negative.

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

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

2.2.2.3.1.2.1.4 Four-Byte Unsigned Encoding (FOUR_BYTE_UNSIGNED_ENCODING)

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



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

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

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

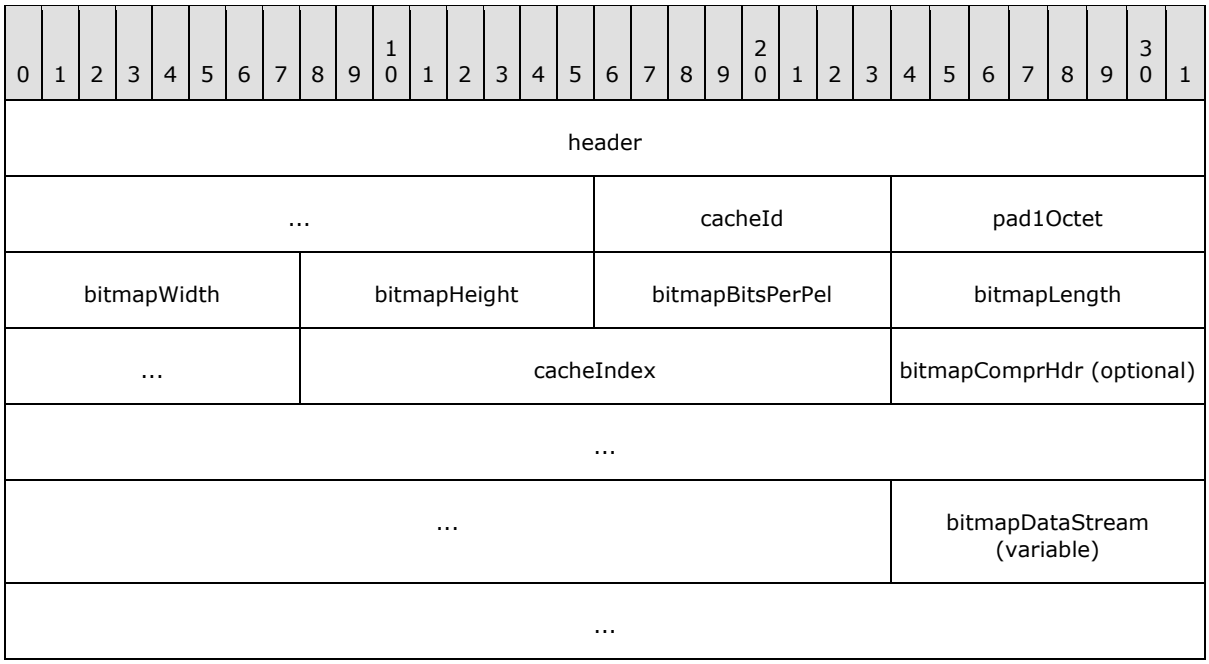
val2 (1 byte): An 8-bit unsigned integer containing the second most significant bits of the value represented by this structure.

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

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

2.2.2.3.1.2.2 Cache Bitmap - Revision 1 (CACHE_BITMAP_ORDER)

The Cache Bitmap - Revision 1 Secondary Drawing Order is used by the server to instruct the client to store a bitmap in a particular Bitmap Cache entry. This order only supports memory-based bitmap caching. Support for Revision 1 bitmap caching is negotiated in the Bitmap Cache Capability Set (Revision 1) (see [MS-RDPBCGR] section 2.2.7.1.5.1).



header (6 bytes): The Secondary Drawing Order Header (section 2.2.2.3.1.2.1.1). The embedded **orderType** field MUST be set to one of the following values:

Value	Meaning
TS_CACHE_BITMAP_UNCOMPRESSED 0x00	The bitmap data in the bitmapDataStream field is uncompressed.
TS_CACHE_BITMAP_COMPRESSED 0x02	The bitmap data in the bitmapDataStream field is compressed.

The embedded **extraFlags** field MAY contain the following flag:

Value	Meaning
NO_BITMAP_COMPRESSION_HDR 0x00000400	Indicates that the **bitmapComprHdr** field is not present (removed for bandwidth efficiency to save 8 bytes).

cacheId (1 byte): An 8-bit unsigned integer. The bitmap cache into which to store the bitmap data. The bitmap cache ID MUST be in the range negotiated by the Bitmap Cache Capability Set (Revision 1) (see [MS-RDPBCGR] section 2.2.7.1.5.1).

pad1Octet (1 byte): An 8-bit unsigned integer. Padding. Values in this field are arbitrary and MUST be ignored.

bitmapWidth (1 byte): An 8-bit unsigned integer. The width of the bitmap in pixels.

bitmapHeight (1 byte): An 8-bit unsigned integer. The height of the bitmap in pixels.

bitmapBitsPerPel (1 byte): An 8-bit unsigned integer. The color depth of the bitmap data in bits-per-pixel. This field MUST be one of the following values:

Value	Meaning
0x08	8-bit color depth.
0x10	16-bit color depth.
0x18	24-bit color depth.
0x20	32-bit color depth.

bitmapLength (2 bytes): A 16-bit unsigned integer. The size in bytes of the data in the **bitmapComprHdr** and **bitmapDataStream** fields.

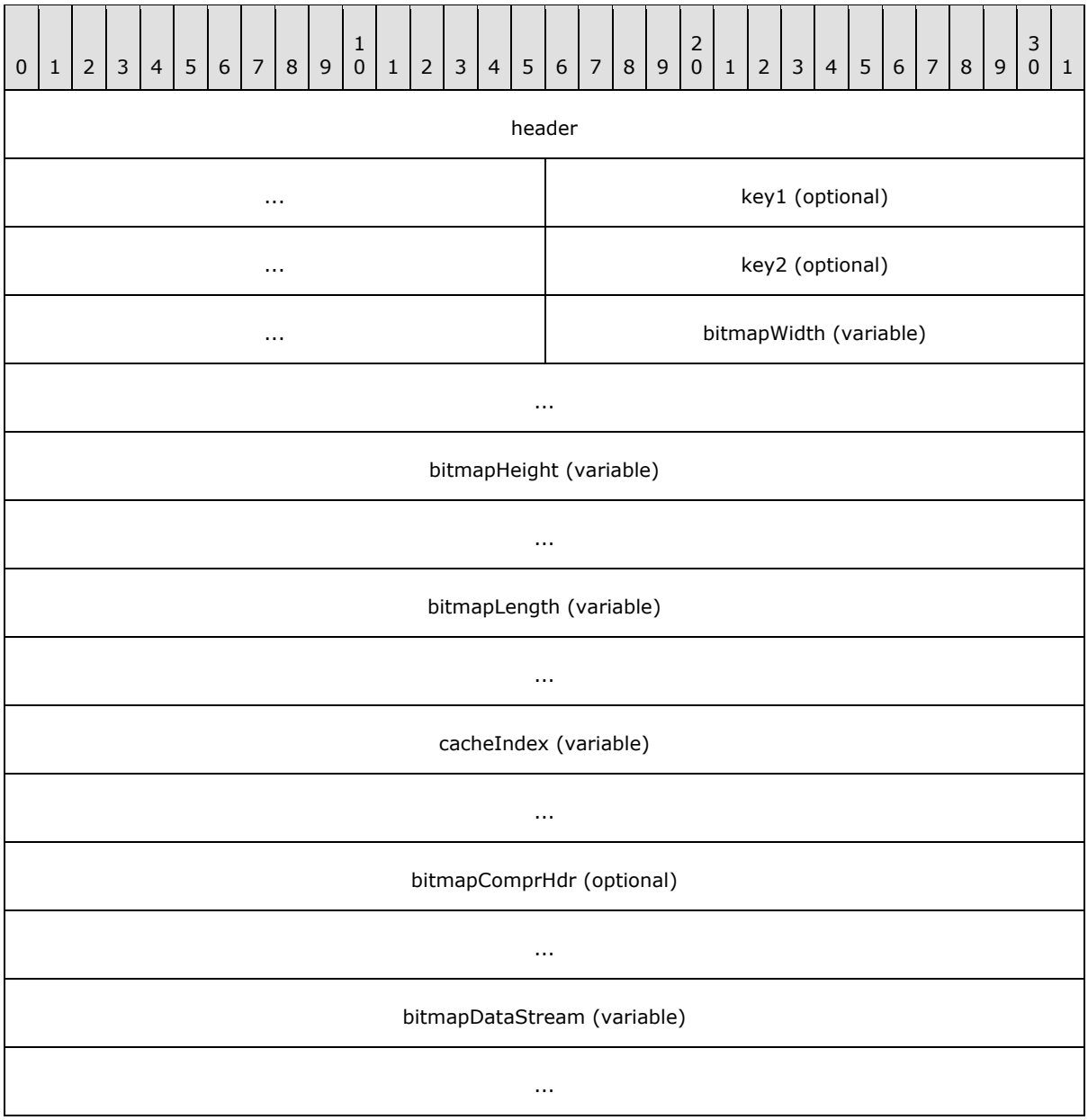
cacheIndex (2 bytes): A 16-bit unsigned integer. An entry in the bitmap cache (specified by the **cacheId** field) where the bitmap should be stored. The bitmap cache index MUST be in the range negotiated by the Bitmap Cache Capability Set (Revision 1) (see [MS-RDPBCGR] section 2.2.7.1.5.1).

bitmapComprHdr (8 bytes): Optional Compressed Data Header structure (see [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.2) describing the bitmap data in the **bitmapDataStream**. This field MUST be present if the TS_CACHE_BITMAP_COMPRESSED (0x02) flag is present in the header field, but the NO_BITMAP_COMPRESSION_HDR (0x0400) flag is not.

bitmapDataStream (variable): A variable-length byte array containing bitmap data (the format of this data is defined in [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.3).

2.2.2.3.1.2.3 Cache Bitmap - Revision 2 (CACHE_BITMAP_REV2_ORDER)

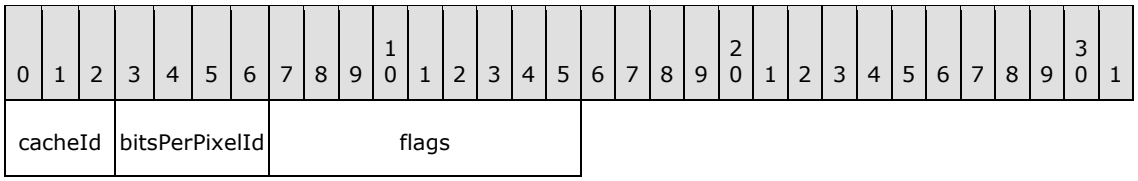
The Cache Bitmap - Revision 2 Secondary Drawing Order is used by the server to instruct the client to store a bitmap in a particular Bitmap Cache entry. This order supports persistent disk bitmap caching and uses a compact encoding format. Support for Revision 2 bitmap caching is negotiated in the Bitmap Cache Capability Set (Revision 2) (see [MS-RDPBCGR] section 2.2.7.1.5.2).



header (6 bytes): A Secondary Drawing Order Header (section 2.2.2.3.1.2.1.1). The embedded **orderType** field MUST be set to one of the following values:

orderType	Meaning
TS_CACHE_BITMAP_UNCOMPRESSED_REV2 0x04	The bitmap data in the **bitmapDataStream** field is uncompressed.
TS_CACHE_BITMAP_COMPRESSED_REV2 0x05	The bitmap data in the **bitmapData** field is compressed.

The format of the embedded **extraFlags** field is specified by the following bitmask diagram:



cacheId (3 bits): A 3-bit unsigned integer. The bitmap cache into which to store the bitmap data. The bitmap cache ID MUST be in the range negotiated by the Bitmap Cache Capability Set (Revision 2) (see [MS-RDPBCGR] section 2.2.7.1.5.2).

bitsPerPixelId (4 bits): A 4-bit unsigned integer. The color depth of the bitmap data in bits-per-pixel. MUST be one of the following values.

Value	Meaning
CBR2_8BPP 0x3	8 bits-per-pixel.
CBR2_16BPP 0x4	16 bits-per-pixel.
CBR2_24BPP 0x5	24 bits-per-pixel.
CBR2_32BPP 0x6	32 bits-per-pixel.

flags (9 bits): A 9-bit unsigned integer. Operational flags.

Value	Meaning
CBR2_HEIGHT_SAME_AS_WIDTH 0x01	Implies that the bitmap height is the same as the bitmap width. If this flag is set, the **bitmapHeight** field MUST NOT be present.
CBR2_PERSISTENT_KEY_PRESENT 0x02	Implies that the bitmap is not intended to be persisted, and the **key1** and **key2** fields MUST NOT be present.
CBR2_NO_BITMAP_COMPRESSION_HDR 0x08	Indicates that the **bitmapComprHdr** field is not present (removed for bandwidth efficiency to save 8 bytes).
CBR2_DO_NOT_CACHE 0x10	Implies that the **cacheIndex** field MUST be ignored, and the bitmap MUST be placed in the last entry of the bitmap cache specified by **cacheId** field.

key1 (4 bytes): A 32-bit unsigned integer. The low 32 bits of the 64-bit persistent bitmap cache key.

key2 (4 bytes): A 32-bit unsigned integer. The high 32 bits of the 64-bit persistent bitmap cache key.

bitmapWidth (variable): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) field. The width of the bitmap in pixels.

bitmapHeight (variable): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) structure. The height of the bitmap in pixels.

bitmapLength (variable): A Four-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.4) structure. The size in bytes of the data in the **bitmapComprHdr** and **bitmapDataStream** fields.

cacheIndex (variable): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) structure. An entry in the bitmap cache (specified by the **cacheId** field) where the bitmap should be stored. The bitmap cache index MUST be in the range negotiated by the Bitmap Cache Capability Set (Revision 2) (see [MS-RDPBCGR] section 2.2.7.1.5.2).

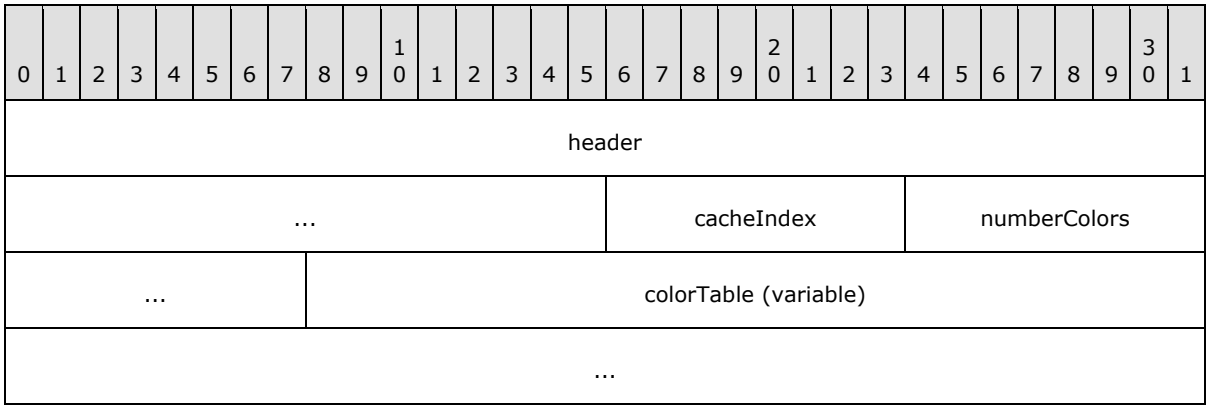
bitmapComprHdr (8 bytes): Optional Compressed Data Header structure (see [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.2) describing the bitmap data in the **bitmapDataStream**. This field MUST be present if the TS_CACHE_BITMAP_COMPRESSED_REV2 (0x05) flag is present in the header field, but the CBR2_NO_BITMAP_COMPRESSION_HDR (0x04) flag is not.

bitmapDataStream (variable): A variable-length byte array containing bitmap data (the format of this data is defined in [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.3).

2.2.2.3.1.2.4 Cache Color Table (CACHE_COLOR_TABLE_ORDER)

The Cache Color Table Secondary Drawing Order is used by the server to instruct the client to store a color table in a particular Color Table Cache entry. Color tables are used in the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders.

Support for color table caching is not negotiated in the Color Table Cache Capability Set (section 2.2.1.1), but is instead implied by support for the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders. If support for these orders is advertised in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3), the existence of a color table cache with entries for six palettes is assumed when palletized color is being used, and the Cache Color Table is used to update these palettes.



header (6 bytes): A Secondary Order Header, as defined in section 2.2.2.3.1.2.1.1. The embedded **orderType** field MUST be set to TS_CACHE_COLOR_TABLE (0x01).

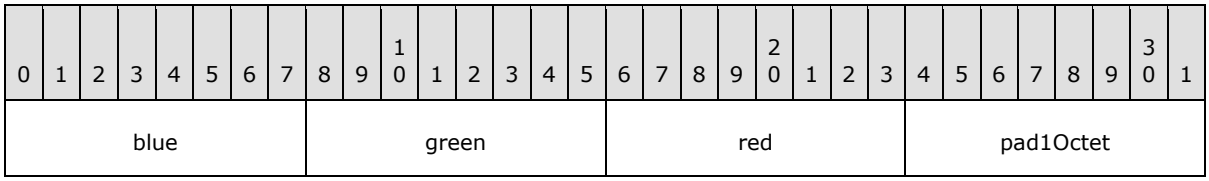
cacheIndex (1 byte): An 8-bit unsigned integer. An entry in the Cache Color Table where the color table MUST be stored. This value MUST be in the range 0 to 5 (inclusive).

numberColors (2 bytes): A 16-bit unsigned integer. The number of Color Quad (section 2.2.2.3.1.2.4.1) structures in the **colorTable** field. This field MUST be set to 256 entries.

colorTable (variable): A Color Table composed of an array of Color Quad (section 2.2.2.3.1.2.4.1) structures. The number of entries in the array is given by the **numberColors** field.

2.2.2.3.1.2.4.1 Color Quad (TS_COLOR_QUAD)

The TS_COLOR_QUAD structure is used to express the red, green, and blue components necessary to reproduce a color in the additive RGB space.



blue (1 byte): An 8-bit unsigned integer. The blue RGB color component.

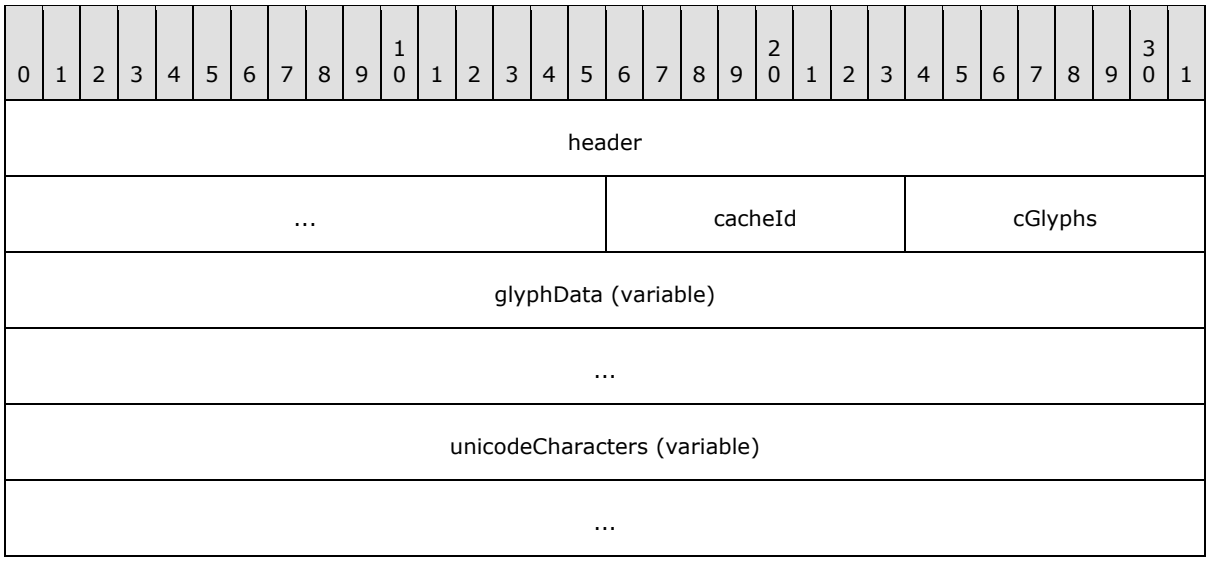
green (1 byte): An 8-bit unsigned integer. The green RGB color component.

red (1 byte): An 8-bit unsigned integer. The red RGB color component.

pad1Octet (1 byte): An 8-bit unsigned integer. Padding. Values in this field are arbitrary and MUST be ignored.

2.2.2.3.1.2.5 Cache Glyph - Revision 1 (CACHE_GLYPH_ORDER)

The Cache Glyph - Revision 1 Secondary Drawing Order is used by the server to instruct the client to store a glyph in a particular Glyph Cache entry. Support for glyph caching is negotiated in the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).



header (6 bytes): A Secondary Order Header, as defined in section 2.2.2.3.1.2.1.1. The embedded **orderType** field MUST be set to TS_CACHE_GLYPH (0x03).

The embedded **extraFlags** field MAY contain the following flags:

Value	Meaning
CG_GLYPH_UNICODE_PRESENT 0x00100	Indicates that the **unicodeCharacters** field is present.

cacheId (1 byte): An 8-bit unsigned integer. The glyph cache into which to store the glyph data. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

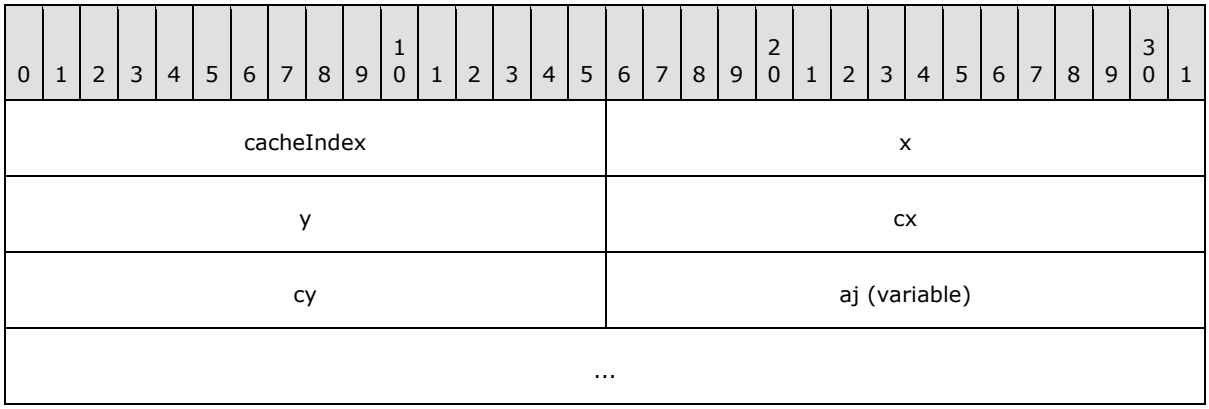
cGlyphs (1 byte): An 8-bit unsigned integer. The number of glyph entries in the **glyphData** field.

glyphData (variable): The specification for each of the glyphs in this order (the number of glyphs is specified by the **cGlyphs** field) defined using Cache Glyph Data structures.

unicodeCharacters (variable): Contains the Unicode character representation of each glyph in the **glyphData** field. The number of bytes in the field is given by **cGlyphs** * 2. This string is used for diagnostic purposes only, and is not necessary for successfully decoding and caching the glyphs in the **glyphData** field.

2.2.2.3.1.2.5.1 Cache Glyph Data (TS_CACHE_GLYPH_DATA)

The TS_CACHE_GLYPH_DATA structure contains information describing a single glyph.



cacheIndex (2 bytes): A 16-bit unsigned integer. The index within a specified Glyph Cache where the glyph data MUST be stored. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

x (2 bytes): A 16-bit signed integer. The X component of the coordinate that defines the origin of the character within the glyph bitmap. The top-left corner of the bitmap is (0, 0).

y (2 bytes): A 16-bit signed integer. The Y component of the coordinate that defines the origin of the character within the glyph bitmap. The top-left corner of the bitmap is (0, 0).

cx (2 bytes): A 16-bit unsigned integer. The width of the glyph bitmap in pixels.

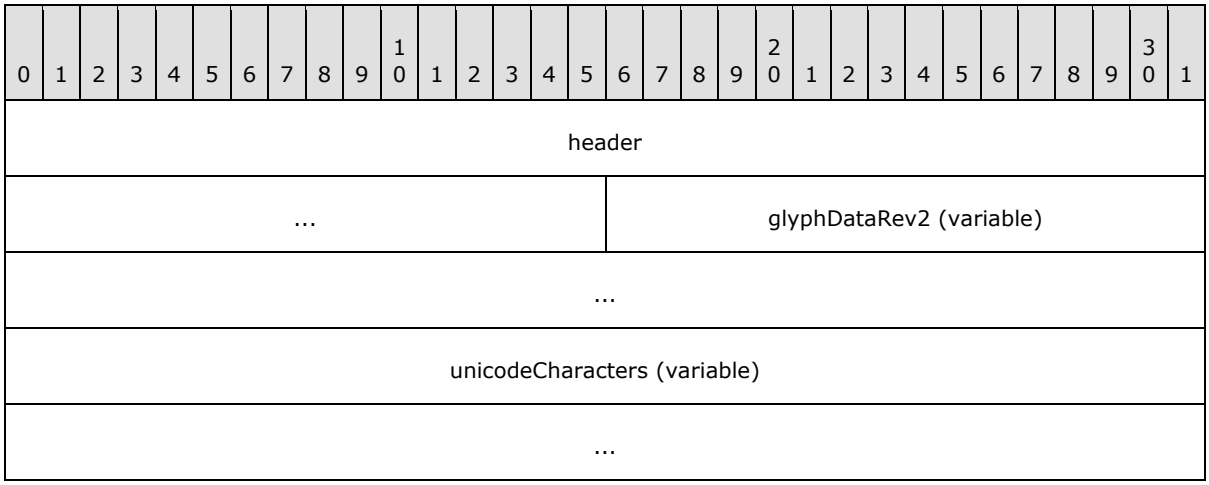
cy (2 bytes): A 16-bit unsigned integer. The height of the glyph bitmap in pixels.

aj (variable): A variable sized byte array containing a 1 bit-per-pixel bitmap of the glyph. The individual scan lines are encoded in top-down order, and each scan line MUST be byte-aligned. Once the array has been populated with bitmap data, it MUST be padded to a double-word

boundary (the size of the structure in bytes MUST be a multiple of 4). For examples of 1 bit-per-pixel encoded glyph bitmaps, see sections 4.8.1 and 4.8.2.

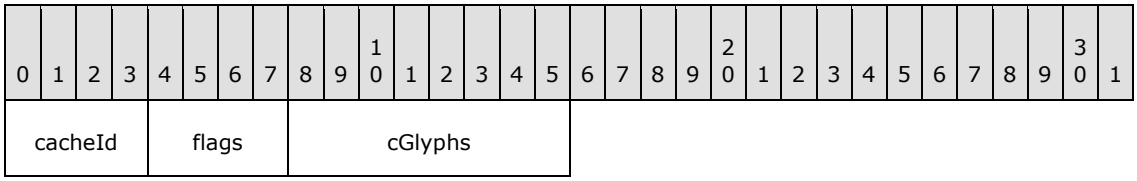
2.2.2.3.1.2.6 Cache Glyph - Revision 2 (CACHE_GLYPH_REV2_ORDER)

The Cache Glyph - Revision 2 Secondary Drawing Order is used by the server to instruct the client to store a glyph in a particular Glyph Cache entry. This order is similar to the Cache Glyph - Revision 1 (section 2.2.2.3.1.2.5) Secondary Drawing Order except that it represents glyphs using a more compact format and moves a number of fields into the **extraFlags** field of the secondary order header. Support for glyph caching is negotiated in the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).



header (6 bytes): A Secondary Order Header, as defined in section 2.2.2.3.1.2.1.1. The embedded **orderType** field MUST be set to TS_CACHE_GLYPH (0x03).

The format of the embedded **extraFlags** word is described by the following bitmask diagram:



cacheId (4 bits): A 4-bit unsigned integer. The glyph cache into which to store the glyph data. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

flags (4 bits): A 4-bit unsigned integer. Various operational flags.

Value	Meaning
CG2_GLYPH_UNICODE_PRESENT 0x01	Indicates that the **unicodeCharacters** field is present.
GLYPH_ORDER_REV2	Indicates that this is a Cache Glyph - Revision 2 Order;

Value	Meaning
0x02	the Cache Glyph - Revision 1 (section 2.2.2.3.1.2.5) **extraFlags** header field does not contain a flag with this value.

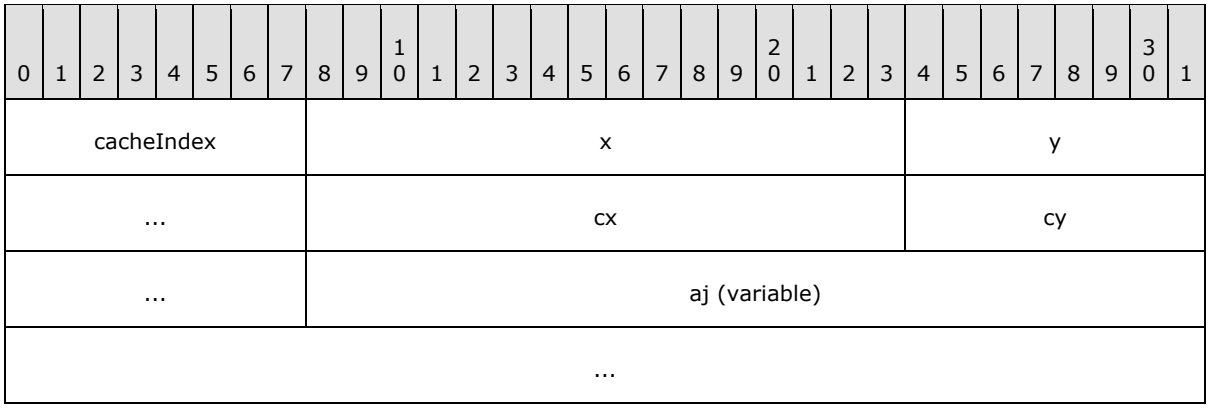
cGlyphs (1 byte): An 8-bit unsigned integer. The number of glyph entries in the **glyphData** field.

glyphDataRev2 (variable): The specification for each of the glyphs in this order (the number of glyphs is given by the **cGlyphs** field embedded in the **header** field) defined using Cache Glyph Data - Revision 2 (section 2.2.2.3.1.2.6.1) structures.

unicodeCharacters (variable): An array of Unicode characters. Contains the Unicode character representation of each glyph in the **glyphData** field. The number of bytes in the field is given by **cGlyphs** * 2 (where **cGlyphs** is embedded in the **header** field). This string is used for diagnostic purposes only, and is not necessary for successfully decoding and caching the glyphs in the **glyphData** field.

2.2.2.3.1.2.6.1 Cache Glyph Data - Revision 2 (TS_CACHE_GLYPH_DATA_REV2)

The TS_CACHE_GLYPH_DATA_REV2 structure contains information describing a single glyph.



cacheIndex (1 byte): An 8-bit unsigned integer. The index within a specified Glyph Cache where the glyph data MUST be stored. This value MUST be in the range negotiated by the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

x (2 bytes): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) structure. The X component of the coordinate that defines the origin of the character within the glyph bitmap. The top-left corner of the bitmap is (0, 0).

y (2 bytes): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) structure. The Y component of the coordinate that defines the origin of the character within the glyph bitmap. The top-left corner of the bitmap is (0, 0).

cx (2 bytes): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) structure. The width of the glyph bitmap in pixels.

cy (2 bytes): A Two-Byte Unsigned Encoding (section 2.2.2.3.1.2.1.2) structure. The height of the glyph bitmap in pixels.

aj (variable): A variable-sized byte array containing a 1 bit-per-pixel bitmap of the glyph.

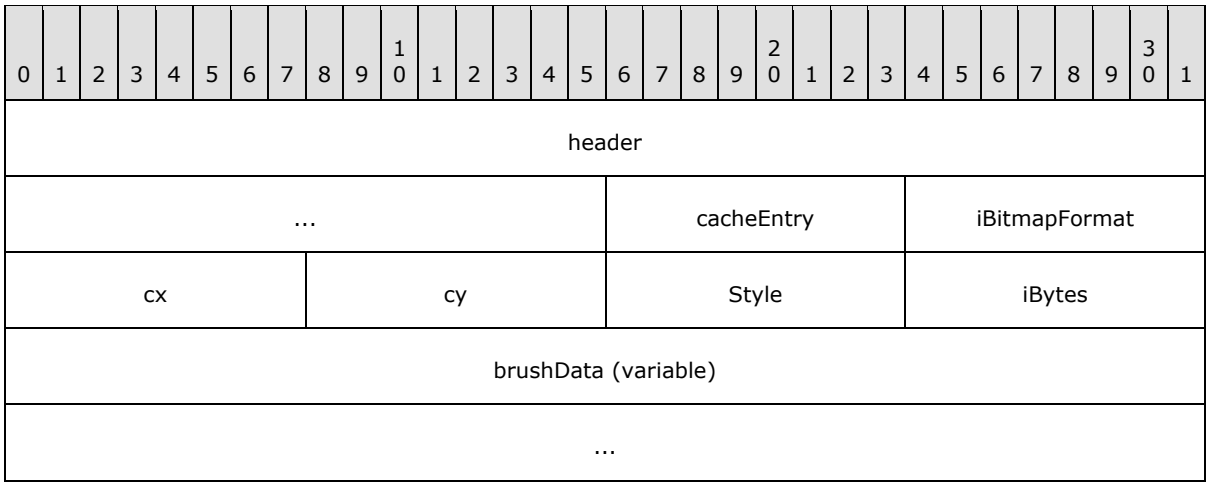
The individual scan lines are encoded in top-down order, and each scan line MUST be byte-aligned. Once the array has been populated with bitmap data, it MUST be padded to a double-word boundary (the size of the structure in bytes MUST be a multiple of 4). The size in bytes of the glyph data is given by the following function:

```
((cx + 7) / 8) * cy
```

For examples of 1 bit-per-pixel encoded glyph bitmaps, see sections 4.8.1 and 4.8.2.

2.2.2.3.1.2.7 Cache Brush (CACHE_BRUSH_ORDER)

The Cache Brush Secondary Drawing Order is used by the server to instruct the client to store a brush in a particular Brush Cache entry. Support for brush caching is negotiated in the Brush Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8).



header (6 bytes): Secondary Order Header, as defined in section 2.2.2.3.1.2.1.1. The embedded **orderType** field MUST be set to TS_CACHE_BRUSH (0x07).

cacheEntry (1 byte): An 8-bit unsigned integer. The entry in a specified Brush Cache where the brush data MUST be stored. This value MUST be in the range 0 to 63 (inclusive).

iBitmapFormat (1 byte): An 8-bit unsigned integer. The color depth of the brush bitmap data. MUST be one of the following values.

Value	Meaning
BMF_1BPP 0x01	1 bit-per-pixel
BMF_8BPP 0x03	8 bits-per-pixel
BMF_16BPP 0x04	15 or 16 bits-per-pixel

Value	Meaning
BMF_24BPP 0x05	24 bits-per-pixel

cx (1 byte): An 8-bit unsigned integer. The width of the brush bitmap.

cy (1 byte): An 8-bit unsigned integer. The height of the brush bitmap.

Style (1 byte): An 8-bit unsigned integer. This field is not used, and SHOULD<4> be set to 0x00.

iBytes (1 byte): An 8-bit unsigned integer. The size of the **brushData** field in bytes.

brushData (variable): A variable sized byte array containing binary brush data that represents an 8 pixel by 8 pixel bitmap image. There are 64 pixels in a brush bitmap, and the space used to represent each pixel depends on the color depth of the brush bitmap and the number of colors used. The size of the **brushData** field in bytes is given by the **iBytes** field.

In general, most brushes only use two colors (mono format), and the majority of the remaining ones use four colors or less.

For mono format brushes (**iBitmapFormat** is BMF_1BPP) **brushData** contains 8 bytes of 1 bit-per-pixel data, each byte corresponding to a row of pixels in the brush. The rows are encoded in reverse order, that is, the pixels in the bottom row of the brush are encoded in the first byte of the **brushData** field, and the pixels in the top row are encoded in the eighth byte.

For four-color brushes, a compression algorithm is used. If the data is compressed, the **iBytes** field is 20 for 256 color (**iBitmapFormat** is BMF_8BPP), 24 for 16-bit color (**iBitmapFormat** is BMF_16BPP), and 28 for 24-bit color (**iBitmapFormat** is BMF_24BPP). The compression algorithm reduces brush data size by storing each brush pixel as a two-bit index (four possible values) into a translation table containing four entries. This equates to two bytes per brush bitmap line (16 bytes in total) followed by the translation table contents. This layout for four-color brushes conforms to the Compressed Color Brush (section 2.2.2.3.1.2.7.1) structure.

For brushes using more than four colors, the data is simply copied uncompressed into the **brushData** at the appropriate color depth.

2.2.2.3.1.2.7.1 Compressed Color Brush (COMPRESSED_COLOR_BRUSH)

The COMPRESSED_COLOR_BRUSH structure is used to hold a compressed version of a four-color 8 pixel by 8 pixel brush.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
p4		p3		p2		p1		p8		p7		p6		p5		p12		p11		p10		p9		p16		p15		p14		p13	
p20		p19		p18		p17		p24		p23		p22		p21		p28		p27		p26		p25		p32		p31		p30		p29	
p36		p35		p34		p33		p40		p39		p38		p37		p44		p43		p42		p41		p48		p47		p46		p45	
p52		p51		p50		p49		p54		p55		p54		p53		p60		p59		p58		p57		p64		p63		p62		p61	
color1 (variable)								color2 (variable)								color3 (variable)								color4 (variable)							

pN: A 2-bit unsigned integer field. The two bit value indicating the translation table entry to use for pixel N.

Value	Meaning
0	Use the first value in the translation table (color1).
1	Use the second value in the translation table (color2).
2	Use the third value in the translation table (color3).
3	Use the fourth value in the translation table (color4).

colorN: Translation table entry N. This entry is an index into the current palette or an RGB triplet value; the actual interpretation depends on the color depth of the bitmap data.

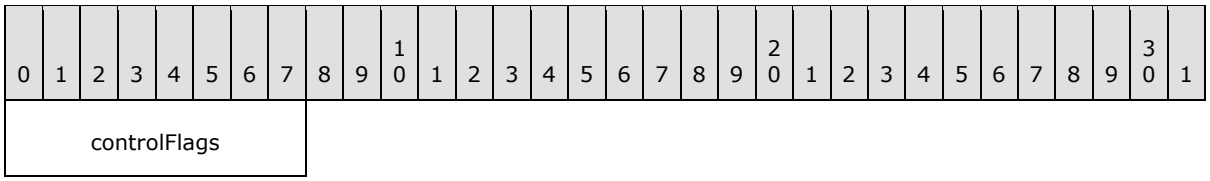
Color depth	Field size	Meaning
8 bpp	1 byte	Index into the current color palette.
15 bpp	2 bytes	RGB color triplet expressed in 5-5-5 format (5 bits for red, 5 bits for green, and 5 bits for blue).
16 bpp	2 bytes	RGB color triplet expressed in 5-6-5 format (5 bits for red, 6 bits for green, and 5 bits for blue).
24 bpp	3 bytes	RGB color triplet (1 byte per component).

2.2.2.3.1.3 Alternate Secondary Drawing Orders

2.2.2.3.1.3.1 Common Data Types

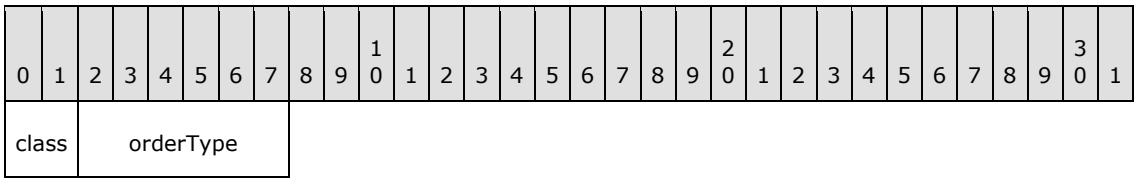
2.2.2.3.1.3.1.1 Alternate Secondary Drawing Order Header (ALTSEC_DRAWING_ORDER_HEADER)

The ALTSEC_DRAWING_ORDER_HEADER structure is included in all alternate secondary drawing orders.



controlFlags (1 byte): An 8-bit unsigned integer. The control byte that identifies the class and type of the drawing order.

The format of the **controlFlags** byte is described by the following bitmask diagram:



class (2 bits): A 2-bit unsigned integer. This field MUST contain only the TS_SECONDARY (0x02) flag to indicate that the order is an alternate secondary drawing order (see section 2.2.2.3.1).

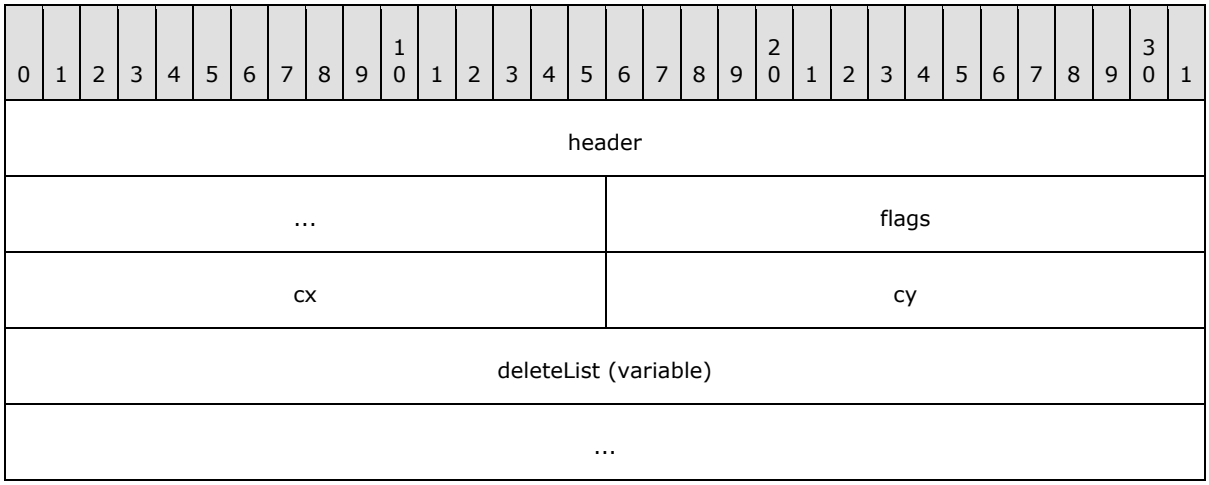
orderType (6 bits): A 6-bit unsigned integer. Identifies the type of alternate secondary drawing order.

Value	Meaning
TS_ALTSEC_SWITCH_SURFACE 0x00	Switch Surface Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.3).
TS_ALTSEC_CREATE_OFFSCR_BITMAP 0x01	Create Offscreen Bitmap Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.2).
TS_ALTSEC_STREAM_BITMAP_FIRST 0x02	Stream Bitmap First (Revision 1 and 2) Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.5.1).
TS_ALTSEC_STREAM_BITMAP_NEXT 0x03	Stream Bitmap Next Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.5.2).
TS_ALTSEC_CREATE_NINEGRID_BITMAP 0x04	Create NineGrid Bitmap Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.4).
TS_ALTSEC_GDIP_FIRST 0x05	Draw GDI+ First Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.6.2).
TS_ALTSEC_GDIP_NEXT 0x06	Draw GDI+ Next Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.6.3).
TS_ALTSEC_GDIP_END 0x07	Draw GDI+ End Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.6.4).
TS_ALTSEC_GDIP_CACHE_FIRST 0x08	Draw GDI+ First Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.6.2).
TS_ALTSEC_GDIP_CACHE_NEXT 0x09	Draw GDI+ Cache Next Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.6.3).

Value	Meaning
TS_ALTSEC_GDIP_CACHE_END 0x0A	Draw GDI+ Cache End Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.6.4).
TS_ALTSEC_WINDOW 0x0B	Windowing Alternate Secondary Drawing Order (see [MS-RDPERP] section 2.2.1.3).
TS_ALTSEC_COMPDESK_FIRST 0x0C	Desktop Composition Alternate Secondary Drawing Order (see [MS-RDPERP] section 2.2).

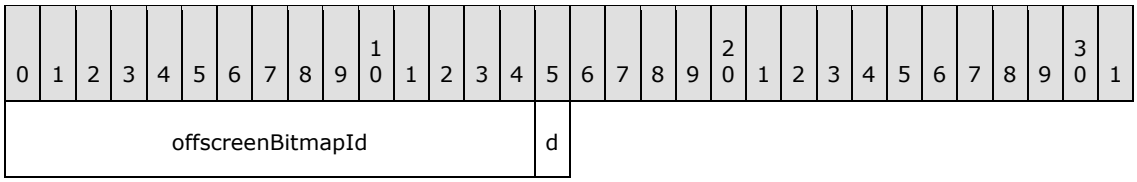
2.2.2.3.1.3.2 Create Offscreen Bitmap (CREATE_OFFSCR_BITMAP_ORDER)

The Create Offscreen Bitmap Alternate Secondary Drawing Order is used by the server to instruct the client to create a bitmap of a particular width and height in the Offscreen Bitmap Cache. Support for offscreen bitmap caching is negotiated in the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10).



header (6 bytes): Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded orderType field MUST be set to TS_ALTSEC_CREATE_OFFSCR_BITMAP (0x01).

flags (2 bytes): A 16-bit unsigned integer. Operational flags. The format of the flags word is described by the following bitmask diagram:



offscreenBitmapId (15 bits): A 15-bit field. The entry in the Offscreen Bitmap Cache where the bitmap MUST be created. This value MUST be in the range negotiated by the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10).

d (1 bit): A 1-bit field. Indicates that the **deleteList** field is present.

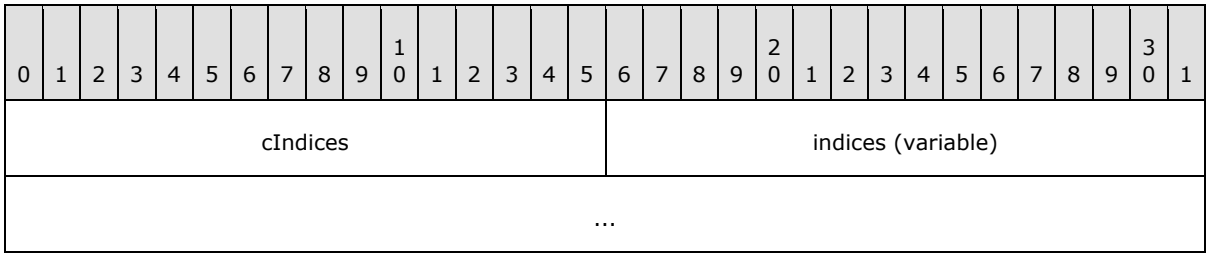
cx (2 bytes): A 16-bit unsigned integer. The width in pixels of the offscreen bitmap to create.

cy (2 bytes): A 16-bit unsigned integer. The height in pixels of the offscreen bitmap to create.

deleteList (variable): A collection of Offscreen Bitmap Cache entries that MUST be deleted, stored in an Offscreen Cache Delete List (section 2.2.2.3.1.3.2.1) structure.

2.2.2.3.1.3.2.1 Offscreen Cache Delete List (OFFSCR_DELETE_LIST)

The OFFSCR_DELETE_LIST structure is used to encode a collection of Offscreen Bitmap Cache indices that MUST be deleted.

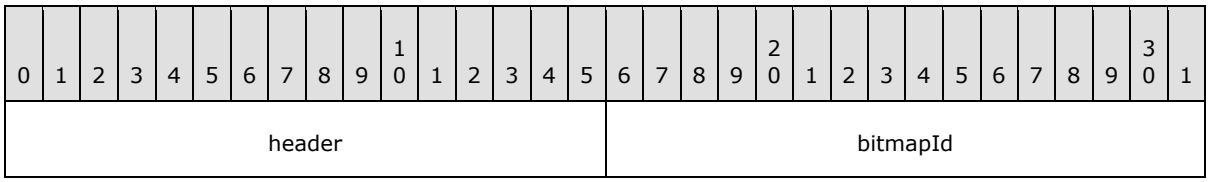


cIndices (2 bytes): A 16-bit unsigned integer. The number of two-byte indices held in the **indices** field.

indices (variable): A collection of offscreen bitmap cache indices that MUST be deleted. Each index is a 16-bit unsigned integer, and MUST be in the range negotiated by the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10). The number of indices in this list is specified by the **cIndices** field.

2.2.2.3.1.3.3 Switch Surface (SWITCH_SURFACE_ORDER)

The Switch Surface Alternate Secondary Drawing Order is used by the server to instruct the client to switch the target drawing surface either to the primary drawing surface (screen desktop) or to an entry in the Offscreen Bitmap Cache. Support for offscreen bitmap caching is negotiated in the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10).



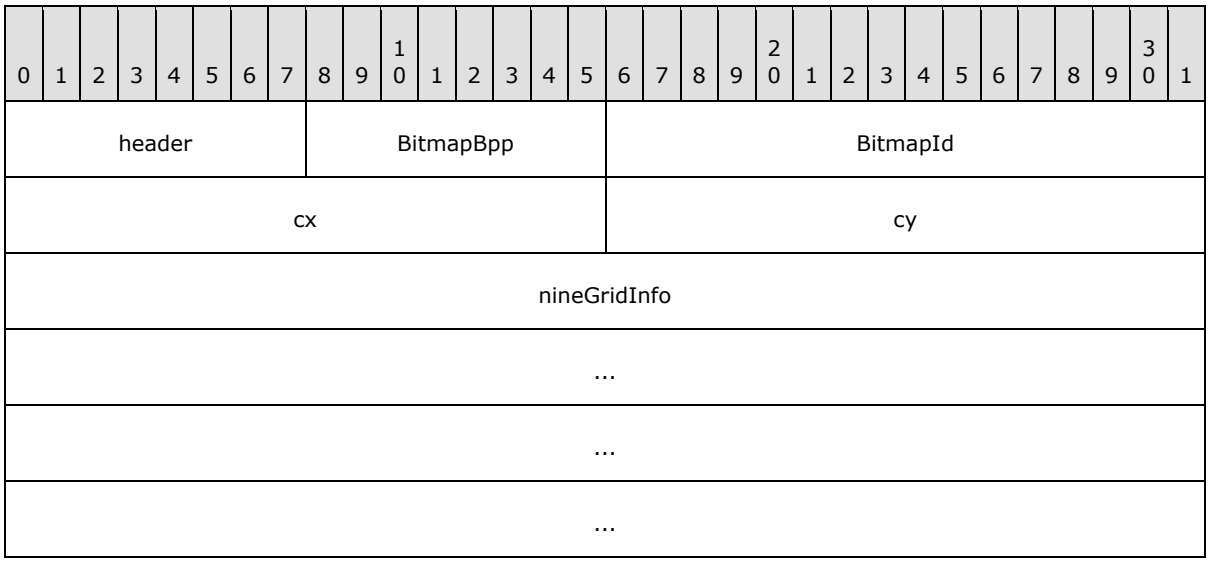
header (2 bytes): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_SWITCH_SURFACE (0x00).

bitmapId (2 bytes): A 16-bit unsigned integer. The new target drawing surface. If this field has a value less than SCREEN_BITMAP_SURFACE (0xFFFF), it identifies a bitmap entry in the Offscreen Bitmap Cache that MUST become the new target drawing surface. This value MUST be in the range negotiated by the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10). Otherwise, if this field has the value SCREEN_BITMAP_SURFACE, the target drawing surface MUST be changed to the primary drawing surface (screen desktop).

2.2.2.3.1.3.4 Create NineGrid Bitmap (CREATE_NINEGRID_BITMAP_ORDER)

The Create NineGrid Bitmap Alternate Secondary Drawing Order is used by the server to instruct the client to create a NineGrid bitmap of a particular width and height in the NineGrid Bitmap Cache (the

color depth MUST be 32 bits-per-pixel). Support for NineGrid drawing is negotiated in the DrawNineGrid Cache Capability Set (section 2.2.1.2).



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_CREATE_NINEGRID_BITMAP (0x04).

BitmapBpp (1 byte): An 8-bit unsigned integer. The color depth in bits-per-pixel of the NineGrid bitmap to create. Currently, all NineGrid bitmaps are sent in 32 bits-per-pixel, so this field MUST be set to 0x20.

BitmapId (2 bytes): A 16-bit unsigned integer. The entry in the NineGrid Bitmap Cache where the bitmap MUST be created. This value MUST be in the range negotiated by the DrawNineGrid Cache Capability Set (section 2.2.1.2).

cx (2 bytes): A 16-bit unsigned integer. The width in pixels of the NineGrid bitmap to create.

cy (2 bytes): A 16-bit unsigned integer. The height in pixels of the NineGrid bitmap to create.

nineGridInfo (16 bytes): A NineGrid Bitmap Information (section 2.2.2.3.1.3.4.1) structure that describes properties of the NineGrid bitmap to be created.

2.2.2.3.1.3.4.1 NineGrid Bitmap Information (NINEGRID_BITMAP_INFO)

The NINEGRID_BITMAP_INFO structure is used to describe a NineGrid source bitmap (see section 4.6). For more information, see [NINEGRID].

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
flFlags																															
ulLeftWidth																				ulRightWidth											
ulTopHeight																				ulBottomHeight											
crTransparent																															

flFlags (4 bytes): A 32-bit unsigned integer. Option flags for the NineGrid bitmap represented by this structure.

Value	Meaning
DSDNG_STRETCH 0x00000001	Indicates that the center portion of the source bitmap MUST be stretched to fill the center of the destination NineGrid.
DSDNG_TILE 0x00000002	Indicates that the center portion source bitmap MUST be tiled to fill the center of the destination NineGrid.
DSDNG_PERPIXELALPHA 0x00000004	Indicates that an AlphaBlend operation (for more information, see [MSDN-ABLEND]) MUST be used to compose the destination NineGrid. The source bitmap is expected to have per-pixel alpha values.
DSDNG_TRANSPARENT 0x00000008	Indicates that a TransparentBlt operation (for more information, see [MSDN-TransparentBlt]) MUST be used to compose the destination NineGrid. The **crTransparent** field MUST contain the transparent color.
DSDNG_MUSTFLIP 0x00000010	Indicates that the source NineGrid MUST be flipped on a vertical axis.
DSDNG_TRUESIZE 0x00000020	Indicates that the source bitmap MUST be transferred without stretching or tiling.

If the DSDNG_TILE (0x00000002) flag is not set, the DSDNG_STRETCH (0x00000001) flag is implied. If neither the DSDNG_PERPIXELALPHA (0x00000004) nor DSDNG_TRANSPARENT (0x00000008) is indicated, a BitBlt operation (for more information, see [MSDN-BitBlt]) MUST be applied.

ulLeftWidth (2 bytes): A 16-bit unsigned integer. The width of the left-side NineGrid border. For a visual illustration of this field, see section 4.6.

ulRightWidth (2 bytes): A 16-bit unsigned integer. The width of the right-side NineGrid border. For a visual illustration of this field, see section 4.6.

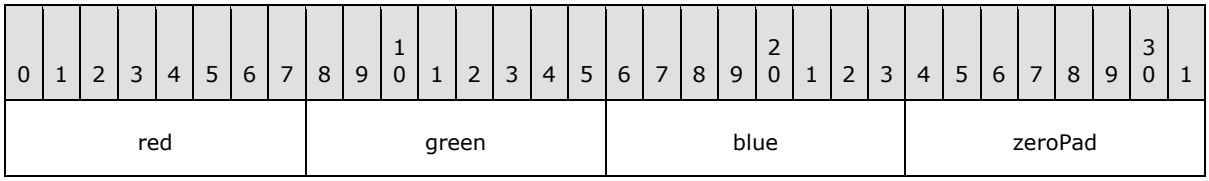
ulTopHeight (2 bytes): A 16-bit unsigned integer. The height of the top NineGrid border. For a visual illustration of this field, see section 4.6.

ulBottomHeight (2 bytes): A 16-bit unsigned integer. The height of the bottom NineGrid border. For a visual illustration of this field, see section 4.6.

crTransparent (4 bytes): The RGB color in the source bitmap to treat as transparent represented using a Color Reference (section 2.2.2.3.1.3.4.1.1) structure. This field is used if the DSDNG_TRANSPARENT (0x00000008) flag is set in the **flFlags** field.

2.2.2.3.1.3.4.1.1 Color Reference (TS_COLORREF)

The TS_COLORREF structure is used to express the red, green, and blue components necessary to reproduce a color in the additive RGB space.



red (1 byte): An 8-bit unsigned integer. The red RGB color component.

green (1 byte): An 8-bit unsigned integer. The green RGB color component.

blue (1 byte): An 8-bit unsigned integer. The blue RGB color component.

zeroPad (1 byte): An 8-bit unsigned integer. Padding. Values in this field MUST be ignored. This field MUST be set to zero.

2.2.2.3.1.3.5 Stream Bitmap Orders

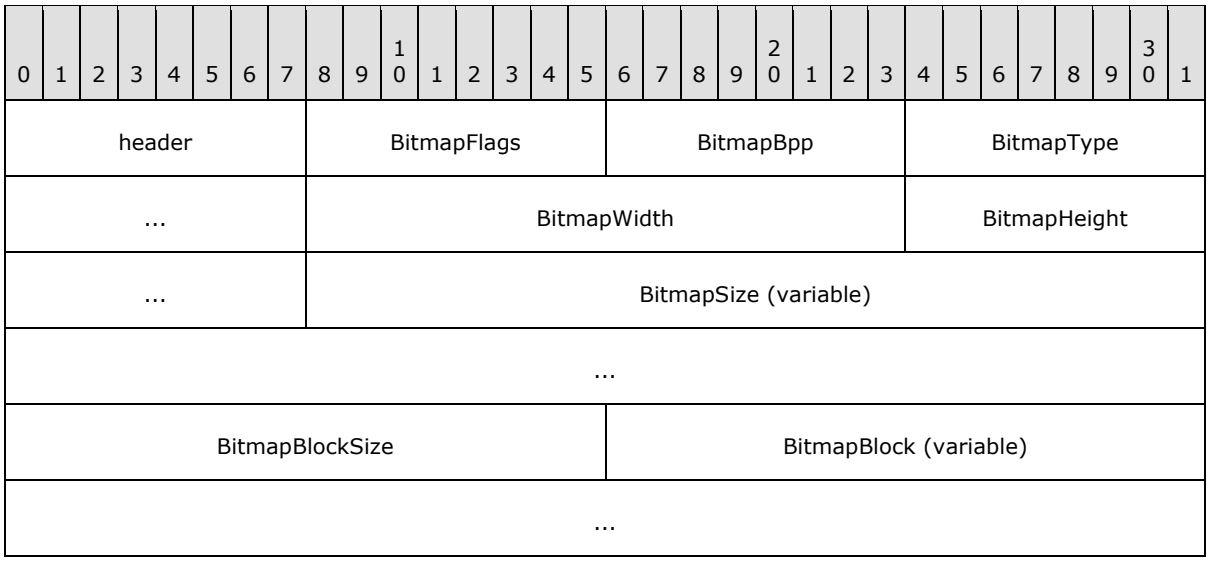
The stream bitmap alternate secondary orders (added in RDP version 5.1) enable bitmaps of arbitrary size to be broken up into 4,096-byte blocks and streamed from server to client, each block being sent in a separate stream bitmap order. All of the bitmap blocks MUST be sent in sequence. The bitmaps can be sent either compressed or uncompressed.

None of the stream bitmap alternate secondary orders include the cache entry in which to store the streamed bitmap. This is because the server MUST always send a bitmap create order that contains the bitmap cache entry data before streaming the bitmap bits to the client.

If support for NineGrid rendering is negotiated using the DrawNineGrid Cache Capability Set (section 2.2.1.2), the support for bitmap streaming is implicitly assumed to be the case, as the NineGrid bitmaps are transported using bitmap streaming.

2.2.2.3.1.3.5.1 Stream Bitmap First (STREAM_BITMAP_FIRST_ORDER)

The Stream Bitmap First Alternate Secondary Drawing Order is used by the server to send the client the first block in a streamed bitmap and information describing the bitmap (such as color depth, width, and height).



header (1 byte): An Alternate Secondary Order Header, as specified in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_STREAM_BITMAP_FIRST (0x02).

BitmapFlags (1 byte): An 8-bit unsigned integer. Flags describing the order contents and layout.

Value	Meaning
STREAM_BITMAP_END 0x01	Indicates that the bitmap fits into one stream bitmap block (4,096 bytes).
STREAM_BITMAP_COMPRESSED 0x02	Indicates that the bitmap data is compressed.
STREAM_BITMAP_REV2 0x04	Indicates that the BitmapSize field is four bytes in size. If this flag is not set, the **BitmapSize** field is two bytes in size.

BitmapBpp (1 byte): An 8-bit unsigned integer. The color depth in bits-per-pixel of the streamed bitmap.

BitmapType (2 bytes): A 16-bit unsigned integer. The type of the streamed bitmap.

Value	Meaning
TS_DRAW_NINEGRID_BITMAP_CACHE 0x0001	Indicates that the data in the **BitmapBlock** field is a NineGrid source bitmap.

BitmapWidth (2 bytes): A 16-bit unsigned integer. The width in pixels of the streamed bitmap.

BitmapHeight (2 bytes): A 16-bit unsigned integer. The height in pixels of the streamed bitmap.

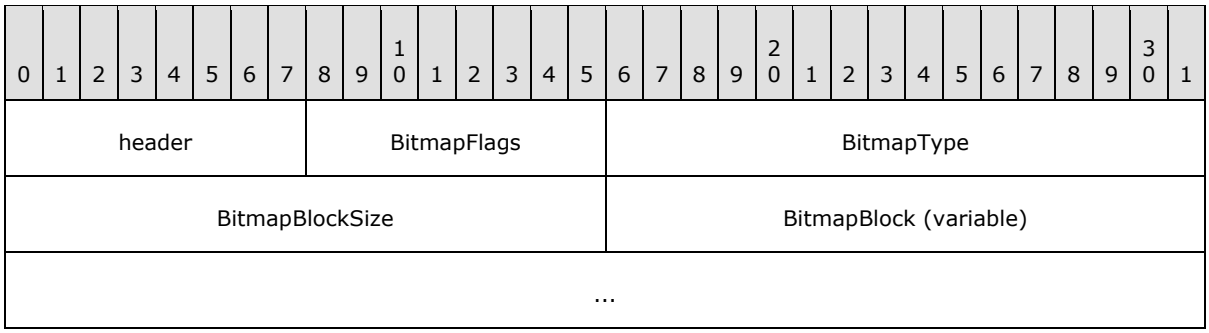
BitmapSize (variable): A variable-length field containing the total size in bytes of the streamed bitmap. If the STREAM_BITMAP_REV2 (0x04) flag is set in the **BitmapFlags** field, this field MUST contain a 32-bit unsigned integer. If the STREAM_BITMAP_REV2 flag is not set, this field MUST contain a 16-bit unsigned integer.

BitmapBlockSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the bitmap stream data block contained in the **BitmapBlock** field. This value MUST be less than or equal to the value contained in the **BitmapSize** field; if the STREAM_BITMAP_END (0x01) flag is set in the **BitmapFlags** field, the two values MUST be equal.

BitmapBlock (variable): A variable-length byte array. The first block of the streamed bitmap (also the last if the STREAM_BITMAP_END (0x01) flag is set in the **BitmapFlags** field). The size of this block is given by the **BitmapBlockSize** field.

2.2.2.3.1.3.5.2 Stream Bitmap Next (STREAM_BITMAP_NEXT_ORDER)

The Stream Bitmap Next Alternate Secondary Drawing Order is used by the server to send the client intermediate and final blocks in a streamed bitmap.



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_STREAM_BITMAP_NEXT (0x03).

BitmapFlags (1 byte): An 8-bit unsigned integer. Flags describing the order contents and layout.

Value	Meaning
STREAM_BITMAP_END 0x01	Indicates that this packet contains the final stream bitmap block.
STREAM_BITMAP_COMPRESSED 0x02	Indicates that the bitmap data is compressed.

BitmapType (2 bytes): A 16-bit unsigned integer. The type of the streamed bitmap.

Value	Meaning
TS_DRAW_NINEGRID_BITMAP_CACHE 0x0001	Indicates that the data in the BitmapBlock field is a NineGrid source bitmap.

BitmapBlockSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the bitmap stream data block contained in the **BitmapBlock** field.

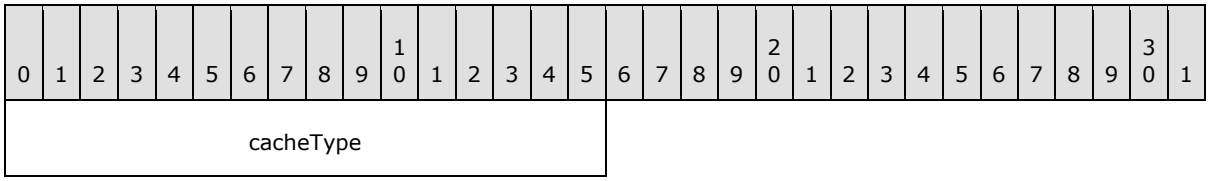
BitmapBlock (variable): A variable-length byte array. The intermediate or final block of the streamed bitmap. The size of this block is given by the **BitmapBlockSize** field.

2.2.2.3.1.3.6 GDI+ Orders

2.2.2.3.1.3.6.1 Common Data Types

2.2.2.3.1.3.6.1.1 GDI+ Cache Type (DRAW_GDIPLUS_CACHE_TYPE)

The DRAW_GDIPLUS_CACHE_TYPE structure stores a GDI+ cache type identifier. The Cache Type is derived from the Type field of an EMF+ Record (see [MS-EMFPLUS] section 2.1.3). Only a subset of these types are cacheable.

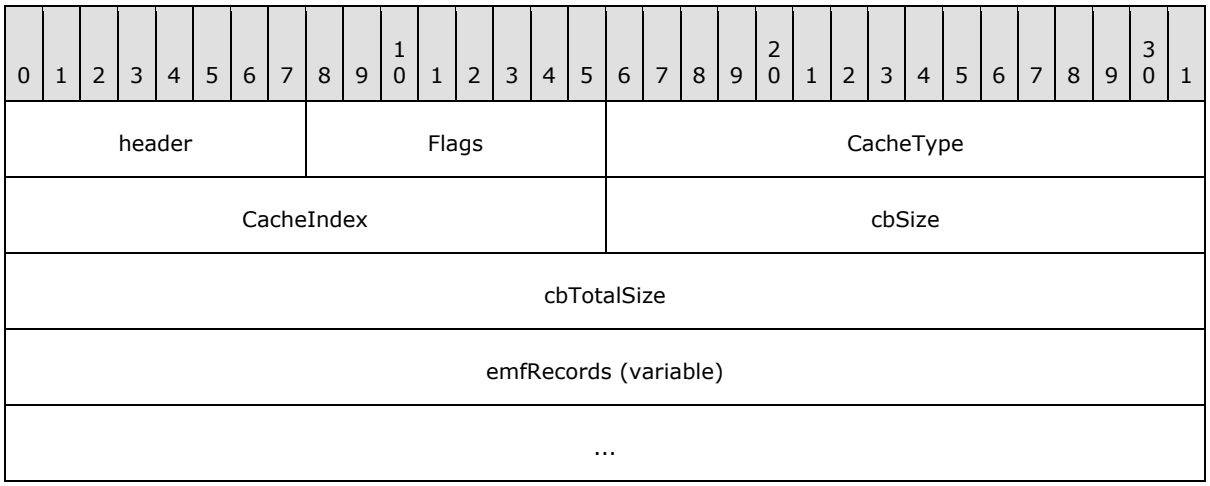


cacheType (2 bytes): A 16-bit unsigned integer. The GDI+ cache type MUST be one of the following values.

Value	Meaning
GDIP_CACHE_GRAPHICS_DATA 0x0001	GDI+ Graphics Cache. This cache type is associated with EMF+ Records of type EmfPlusRecordTypeSetTSGraphics (see [MS-EMFPLUS] section 2.1.3.55).
GDIP_CACHE_OBJECT_BRUSH 0x0002	GDI+ Brush Cache. This cache type is associated with EMF+ Records of type EmfPlusRecordTypeObject (see [MS-EMFPLUS] section 2.1.3.34) where the object type is ObjectTypeBrush (see [MS-EMFPLUS] section **2.1.1.32**).
GDIP_CACHE_OBJECT_PEN 0x0003	GDI+ Pen Cache. This cache type is associated with EMF+ Records of type EmfPlusRecordTypeObject (see [MS-EMFPLUS] section 2.1.3.34) where the object type is ObjectTypePen (see [MS-EMFPLUS] section **2.1.1.32**).
GDIP_CACHE_OBJECT_IMAGE 0x0004	GDI+ Image Cache. This cache type is associated with EMF+ Records of type EmfPlusRecordTypeObject (see [MS-EMFPLUS] section 2.1.3.34) where the object type is ObjectTypeImage (see [MS-EMFPLUS] section **2.1.1.32**).
GDIP_CACHE_OBJECT_IMAGEATTRIBUTES 0x0005	GDI+ Image Attributes Cache. This cache type is associated with EMF+ Records of type EmfPlusRecordTypeObject (see [MS-EMFPLUS] section 2.1.3.34) where the object type is ObjectTypeImageAttributes (see [MS-EMFPLUS] section **2.1.1.32**).

2.2.2.3.1.3.6.2 Draw GDI+ Cache First (DRAW_GDIPLUS_CACHE_FIRST_ORDER)

The Draw GDI+ Cache First Alternate Secondary Drawing Order contains the first batch of GDI+ 1.1 cacheable drawing primitives that comprise a rendering update sent by the server to the client. Support for GDI+ 1.1 rendering is negotiated in the Draw GDI+ Capability Set (section 2.2.1.3).



header (1 byte): An Alternate Secondary Order Header, as specified in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_GDIP_CACHE_FIRST (0x08).

Flags (1 byte): An 8-bit unsigned integer. Flags indicating instructions on how to handle previous cache orders in relation to this one.

Value	Meaning
GDIP_REMOVE_CACHEENTRY 0x01	Remove the cache entry item at the index specified by **CacheIndex** before caching the one contained in this order.

CacheType (2 bytes): A GDI+ Cache Type (section 2.2.2.3.1.3.6.1.1) structure. GDI+ cache in which to store the EMF+ Records contained in the **emfRecords** field.

CacheIndex (2 bytes): A 16-bit unsigned integer. The specific cache entry into which to write the primitive. This value MUST be in the range negotiated by the Draw GDI+ Capability Set (section 2.2.1.3).

cbSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the **emfRecords** field.

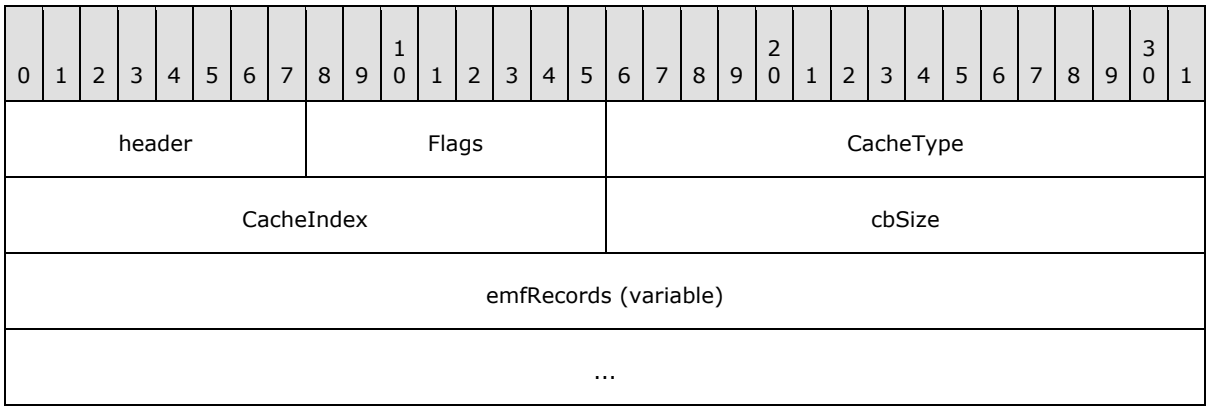
cbTotalSize (4 bytes): A 32-bit unsigned integer. The cumulative size in bytes of the data in all of the **emfRecords** fields in this and subsequent Draw GDI+ Cache Next and Draw GDI+ Cache End Orders.

emfRecords (variable): A collection of EMF+ Records specified in [MS-EMFPLUS] section 2.1.3. The size of the **emfRecords** field is given by the **cbSize** field.

2.2.2.3.1.3.6.3 Draw GDI+ Cache Next (DRAW_GDIPLUS_CACHE_NEXT_ORDER)

The Draw GDI+ Cache Next Alternate Secondary Drawing Order contains the second or subsequent batch of GDI+ 1.1 cacheable drawing primitives that comprise a rendering update sent by the server to the client. The first primitive in the sequence MUST have been transmitted with the Draw GDI+

Cache First Alternate Secondary Drawing Order. Support for GDI+ 1.1 rendering is negotiated in the Draw GDI+ Capability Set (section 2.2.1.3).



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_GDIP_CACHE_NEXT (0x09).

Flags (1 byte): An 8-bit unsigned integer. Flags indicating instructions on how to handle previous cache orders in relation to this one.

Value	Meaning
GDIP_REMOVE_CACHEENTRY 0x01	Remove the cache entry item at the index specified by **CacheIndex** before caching the one contained in this order.

CacheType (2 bytes): A GDI+ Cache Type (section 2.2.2.3.1.3.6.1.1) structure. GDI+ cache in which to store the EMF+ Records contained in the **emfRecords** field.

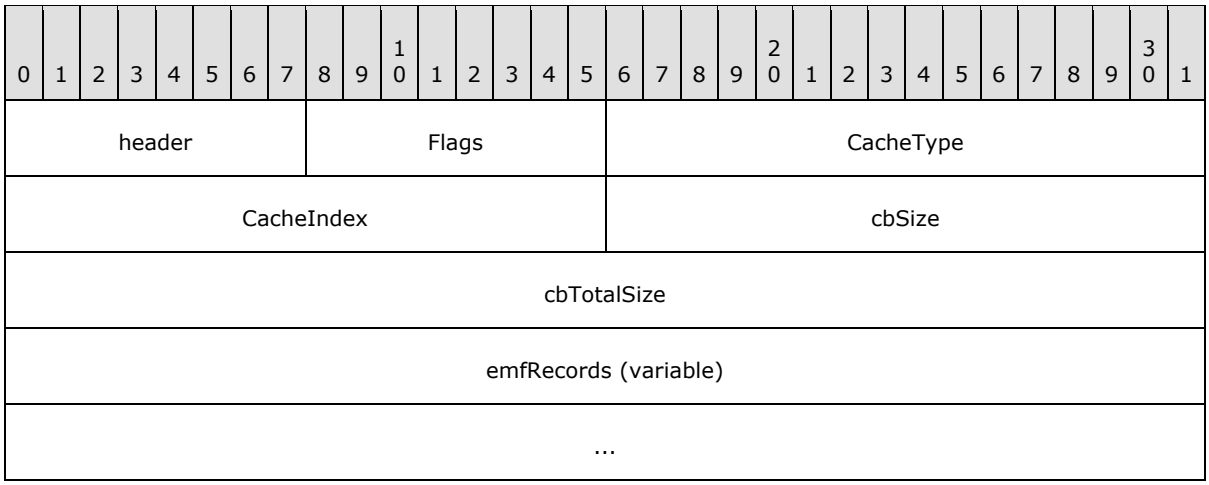
CacheIndex (2 bytes): A 16-bit unsigned integer. The specific cache entry into which to write the primitive. This value MUST be in the range negotiated by the Draw GDI+ Capability Set (section 2.2.1.3).

cbSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the **emfRecords** field.

emfRecords (variable): A collection of EMF+ Records specified in [MS-EMFPLUS] section 2.1.3. The size of the **emfRecords** field is given by the **cbSize** field.

2.2.2.3.1.3.6.4 Draw GDI+ Cache End (DRAW_GDIPLUS_CACHE_END_ORDER)

The Draw GDI+ Cache End Alternate Secondary Drawing Order contains the final batch of GDI+ 1.1 cacheable drawing primitives that comprise a rendering update sent by the server to the client. Support for GDI+ 1.1 rendering is negotiated in the Draw GDI+ Capability Set (section 2.2.1.3).



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_GDIP_CACHE_END (0x0A).

Flags (1 byte): An 8-bit unsigned integer. Flags indicating instructions on how to handle previous cache orders in relation to this one.

Value	Meaning
GDIP_REMOVE_CACHEENTRY 0x01	Remove the cache entry item at the index specified by **CacheIndex** before caching the one contained in this order.

CacheType (2 bytes): A GDI+ Cache Type (section 2.2.2.3.1.3.6.1.1) structure. GDI+ cache in which to store the EMF+ Records contained in the **emfRecords** field.

CacheIndex (2 bytes): A 16-bit unsigned integer. The specific cache entry into which to write the primitive. This value MUST be in the range negotiated by the Draw GDI+ Capability Set (section 2.2.1.3).

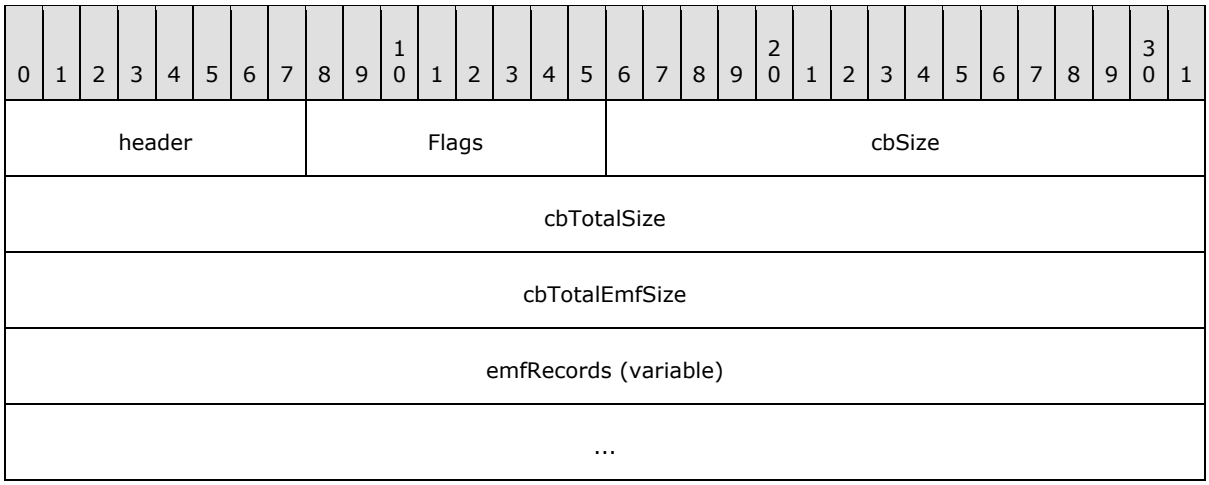
cbSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the **emfRecords** field.

cbTotalSize (4 bytes): A 32-bit unsigned integer. The cumulative size in bytes of the data in all of the **emfRecords** fields in this and previous Draw GDI+ Cache Next and Draw GDI+ Cache End Orders.

emfRecords (variable): A collection of EMF+ Records as defined in Section 2.1.3 of [MS-EMFPLUS]. The size of the **emfRecords** field is given by the **cbSize** field.

2.2.2.3.1.3.6.5 Draw GDI+ First (DRAW_GDIPLUS_FIRST_ORDER)

The Draw GDI+ First Alternate Secondary Drawing Order contains the first batch of GDI+ 1.1 drawing primitives that comprise a rendering update sent by the server to the client. Support for GDI+ 1.1 rendering is negotiated in the Draw GDI+ Capability Set (section 2.2.1.3).



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_GDIP_FIRST (0x05).

Flags (1 byte): An 8-bit unsigned integer. This field is not used, and SHOULD<5> be set to 0x00.

cbSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the **emfRecords** field.

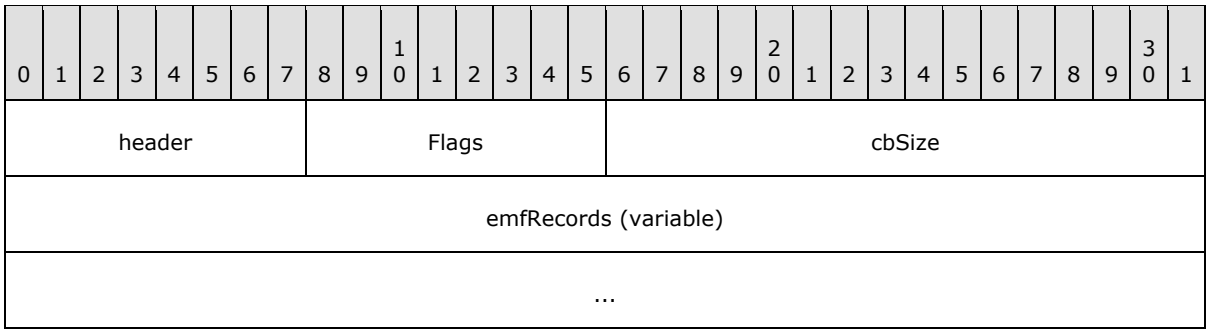
cbTotalSize (4 bytes): A 32-bit unsigned integer. The cumulative size in bytes of the data in all of the **emfRecords** fields in this and subsequent Draw GDI+ Next and Draw GDI+ End orders.

cbTotalEmfSize (4 bytes): A 32-bit unsigned integer. The cumulative size in bytes of the EMF+ record data in this and subsequent Draw GDI+ Next and Draw GDI+ End Orders. The value in the **cbTotalEmfSize** field MUST be smaller than the value in the **cbTotalSize** field, as the actual EMF+ record data is a subset of the data contained in the **emfRecords** field.

emfRecords (variable): A collection of EMF+ Records specified in [MS-EMFPLUS] section 2.1.3. The size of the **emfRecords** field is given by the **cbSize** field. If the most significant bit of the **Size** field of an EMF+ Record is set, the EMF+ Record data contains a 16-bit cache index that MUST be used to retrieve a cached GDI+ 1.1 primitive from the appropriate GDI+ cache.

2.2.2.3.1.3.6.6 Draw GDI+ Next (DRAW_GDIPLUS_NEXT_ORDER)

The Draw GDI+ Next Alternate Secondary Drawing Order contains the second or subsequent batch of GDI+ 1.1 drawing primitives that comprise a rendering update sent by the server to the client. The first primitive in the sequence MUST have been transmitted with the Draw GDI+ First Alternate Secondary Drawing Order. Support for GDI+ 1.1 rendering is negotiated in the Draw GDI+ Capability Set (section 2.2.1.3).



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_GDIP_NEXT (0x06).

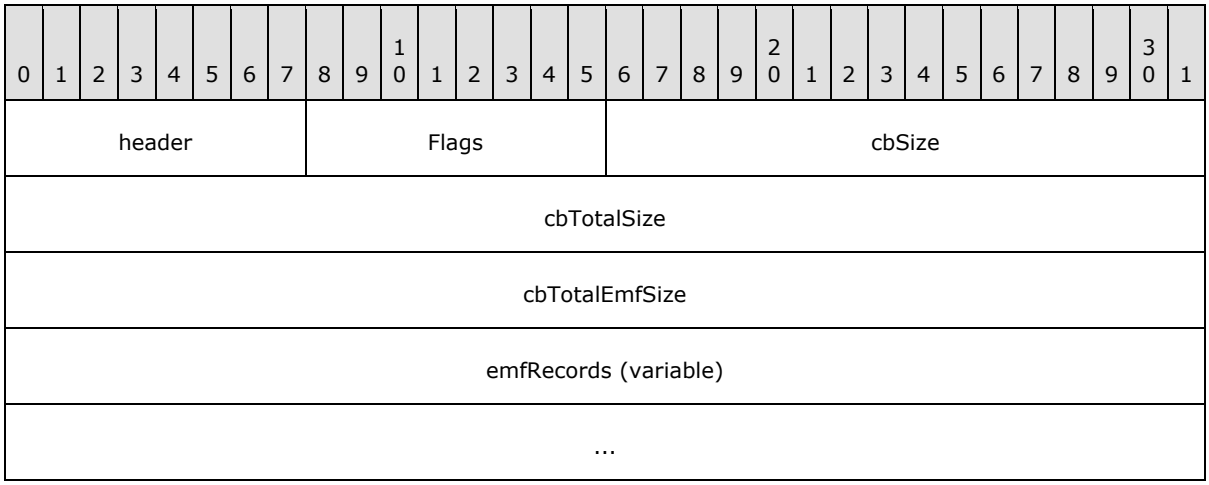
Flags (1 byte): An 8-bit unsigned integer. This field is not used, and SHOULD<6> be set to 0x00.

cbSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the **emfRecords** field.

emfRecords (variable): Collection of EMF+ Records specified in [MS-EMFPLUS] section 2.1.3. The size of the **emfRecords** field is given by the **cbSize** field. If the most significant bit of the **Size** field of an EMF+ Record is set, the EMF+ Record data contains a 16-bit cache index that MUST be used to retrieve a cached GDI+ 1.1 primitive from the appropriate GDI+ cache.

2.2.2.3.1.3.6.7 Draw GDI+ End (DRAW_GDIPLUS_END_ORDER)

The Draw GDI+ End Alternate Secondary Drawing Order contains the final batch of GDI+ 1.1 drawing primitives that comprise a rendering update sent by the server to the client. Support for GDI+ 1.1 rendering is negotiated in the Draw GDI+ Capability Set (section 2.2.1.3).



header (1 byte): An Alternate Secondary Order Header, as defined in section 2.2.2.3.1.3.1.1. The embedded **orderType** field MUST be set to TS_ALTSEC_GDIP_END (0x07).

Flags (1 byte): An 8-bit unsigned integer. This field is not used, and SHOULD<7> be set to 0x00.

cbSize (2 bytes): A 16-bit unsigned integer. The size in bytes of the **emfRecords** field.

cbTotalSize (4 bytes): A 32-bit unsigned integer. The cumulative size in bytes of the data in all of the **emfRecords** fields in this and previous Draw GDI+ First and Draw GDI+ Next Orders.

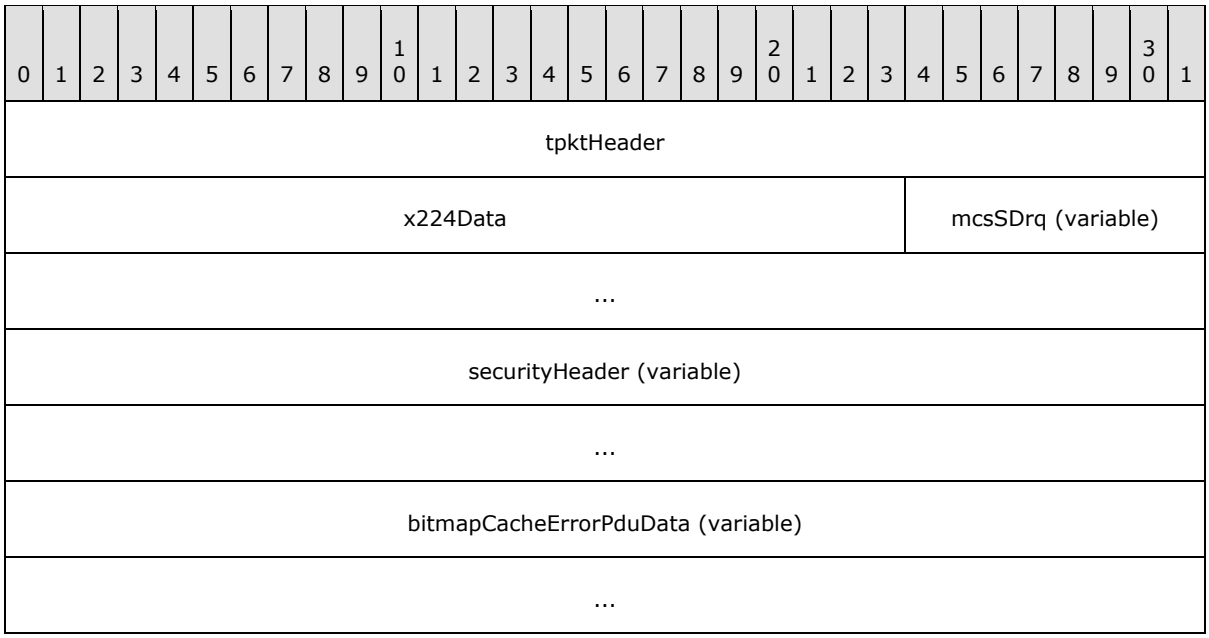
cbTotalEmfSize (4 bytes): A 32-bit unsigned integer. The cumulative size in bytes of the EMF+ record data in this and previous Draw GDI+ First and Draw GDI+ Next Orders. The value in the **cbTotalEmfSize** field MUST be smaller than the value in the **cbTotalSize** field, as the actual EMF+ record data is a subset of the data contained in the **emfRecords** field.

emfRecords (variable): A collection of EMF+ Records specified in [MS-EMFPLUS] section 2.1.3. The size of the **emfRecords** field is given by the **cbSize** field. If the most significant bit of the **Size** field of an EMF+ Record is set, the EMF+ Record data contains a 16-bit cache index that MUST be used to retrieve a cached GDI+ 1.1 primitive from the appropriate GDI+ cache.

2.2.2.4 Error Conditions

2.2.2.4.1 Client Bitmap Cache Error PDU

The Bitmap Cache Error PDU is sent by clients that support persistent bitmap caching and encounter caching errors. The server will honor up to five Bitmap Cache Error PDUs in a session; further error PDUs are ignored to reduce server overhead.



tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable-length MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is specified in [T125] part 7 section 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Bitmap Cache Error PDU data.

securityHeader (variable): Optional security header. If Standard RDP Security is in effect, and the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and

2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), this field will contain one of the following headers:

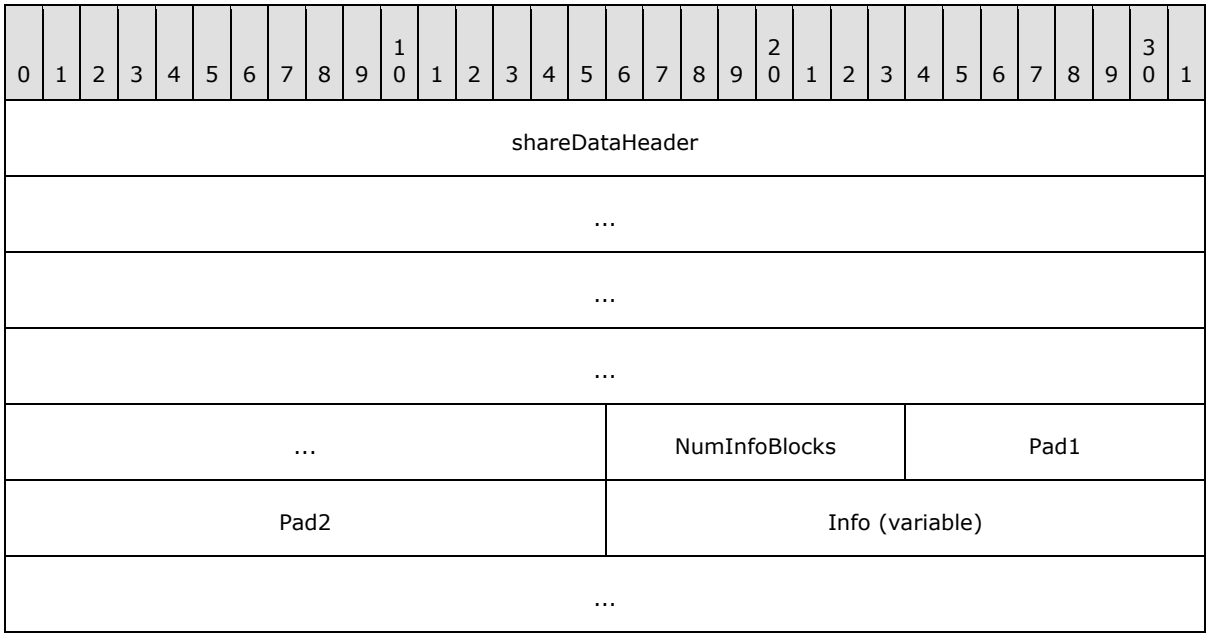
- Non-FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2), or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is in effect, or the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), this header is not included in the PDU.

bitmapCacheErrorPduData (variable): The actual contents of the Bitmap Cache Error PDU, as specified in section 2.2.2.4.1.1.

2.2.2.4.1.1 Bitmap Cache Error PDU Data (TS_BITMAP_CACHE_ERROR_PDU)

The TS_BITMAP_CACHE_ERROR_PDU structure contains the contents of the Bitmap Cache Error PDU, which is essentially a Share Data Header (see [MS-RDPBCGR] section 2.2.8.1.1.1.2) and an array of Bitmap Cache Error Info (section 2.2.2.4.1.1.1) structures.



shareDataHeader (18 bytes): A Share Data Header (as specified in [MS-RDPBCGR] section 2.2.8.1.1.1.2) containing information on the packet. The type subfield of the **pduType** field of the Share Control Header within the Share Data Header MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_BITMAPCACHE_ERROR_PDU (44).

NumInfoBlocks (1 byte): An 8-bit unsigned integer. The number of Bitmap Cache Error Info (section 2.2.2.4.1.1.1) structures in the **Info** field.

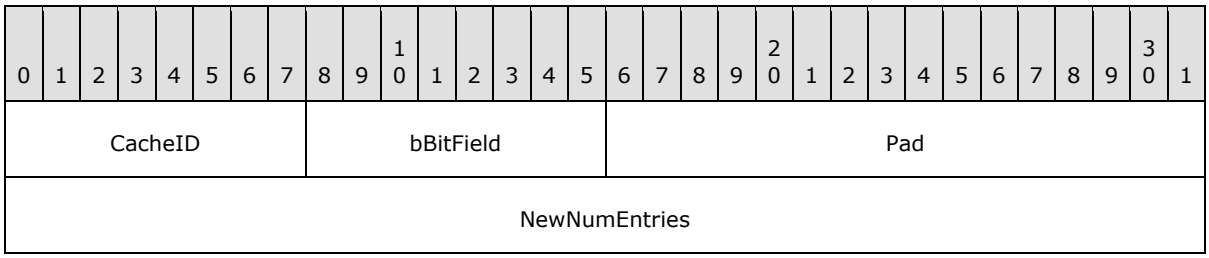
Pad1 (1 byte): An 8-bit unsigned integer. Padding. Values in this field are arbitrary, and MUST be ignored.

Pad2 (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are arbitrary, and MUST be ignored.

Info (variable): An array of Bitmap Cache Error Info (section 2.2.2.4.1.1.1) structures, each structure specifying what actions to take with each of the bitmap caches.

2.2.2.4.1.1.1 Bitmap Cache Error Info (TS_BITMAP_CACHE_ERROR_INFO)

The TS_BITMAP_CACHE_ERROR_INFO structure specifies what actions are to be taken on a particular bitmap cache when a caching error occurs.



CacheID (1 byte): An 8-bit unsigned integer. ID of the bitmap cache represented by this block.

bBitField (1 byte): An 8-bit unsigned integer. A bit field containing several flags.

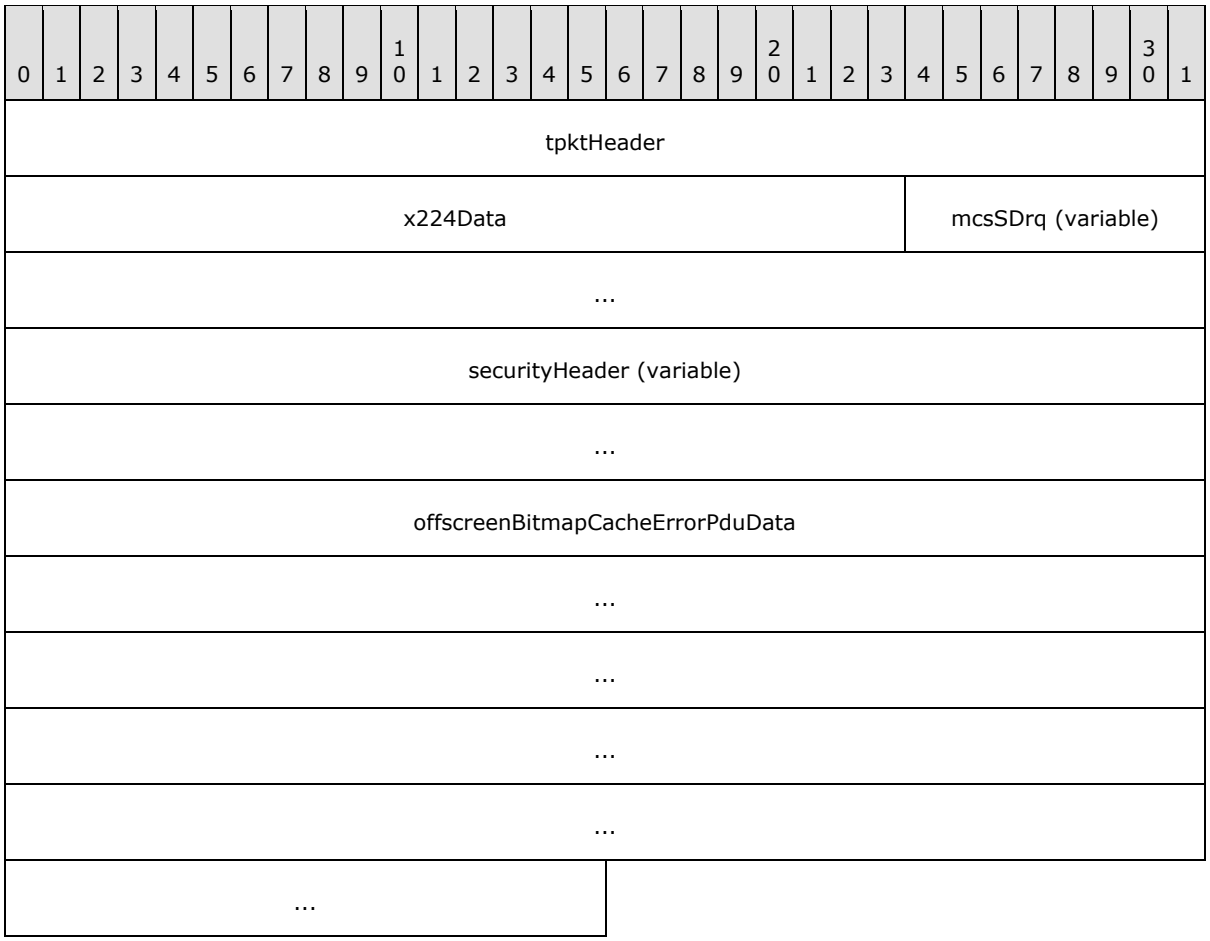
Value	Meaning
BC_ERR_FLUSH_CACHE 0x01	Indicates that the contents of the cache MUST be emptied.
BC_ERR_NEWNUMENTRIES_VALID 0x02	Indicates that the **NewNumEntries** field is valid. If the BC_ERR_FLUSH_CACHE (0x01) flag is not set, and the **NewNumEntries** field specifies a new non-zero size, the previous cache contents in the initial **NewNumEntries** cells MUST be preserved.

Pad (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are ignored.

NewNumEntries (4 bytes): A 32-bit unsigned integer. The new number of entries in the cache. This value must be less than or equal to the number of entries specified in the (Revision 2) Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.2).

2.2.2.4.2 Client Offscreen Bitmap Cache Error PDU

The Offscreen Bitmap Cache Error PDU is sent by clients that support offscreen bitmap caching and encounter caching errors.



tpktHeader (4 bytes): A TPKT Header as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable-length MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is specified in [T125] part 7 section 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the Offscreen Bitmap Cache Error PDU data.

securityHeader (variable): An optional security header. If Standard RDP Security is in effect, and the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), this field will contain one of the following headers:

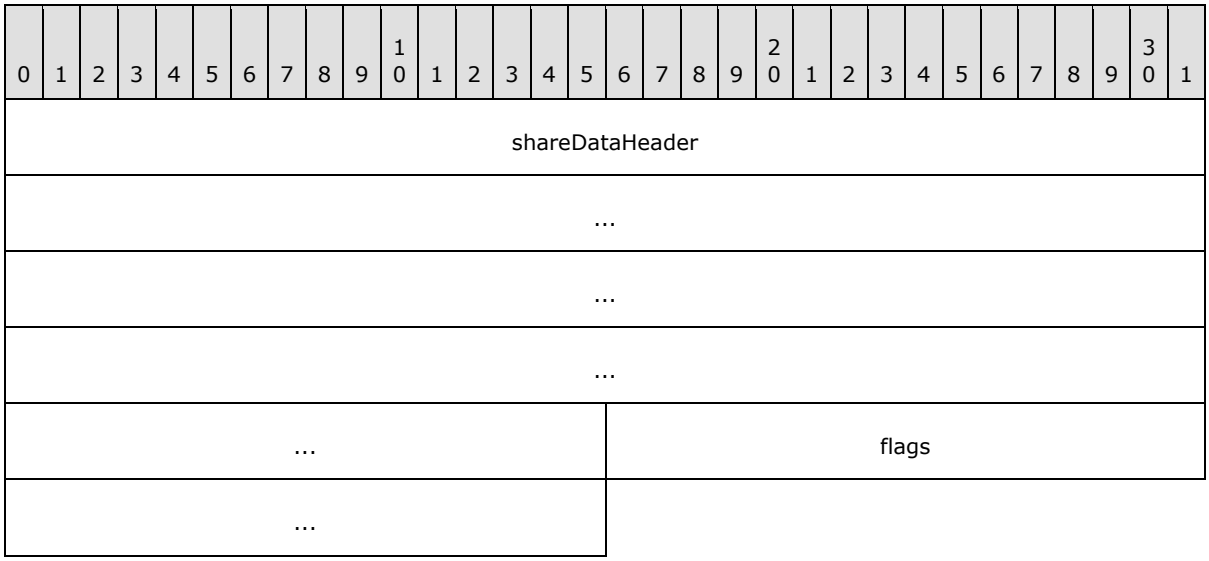
- Non-FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2), or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is in effect, or the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), this header is not included in the PDU.

offscreenBitmapCacheErrorPduData (22 bytes): The actual contents of the Offscreen Bitmap Cache Error PDU, as specified in section 2.2.2.4.2.1.

2.2.2.4.2.1 Offscreen Bitmap Cache Error PDU Data (TS_OFFSCRCACHE_ERROR_PDU)

The TS_OFFSCRCACHE_ERROR_PDU structure contains the contents of the Offscreen Bitmap Cache Error PDU, which is essentially a Share Data Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) and a flags field.



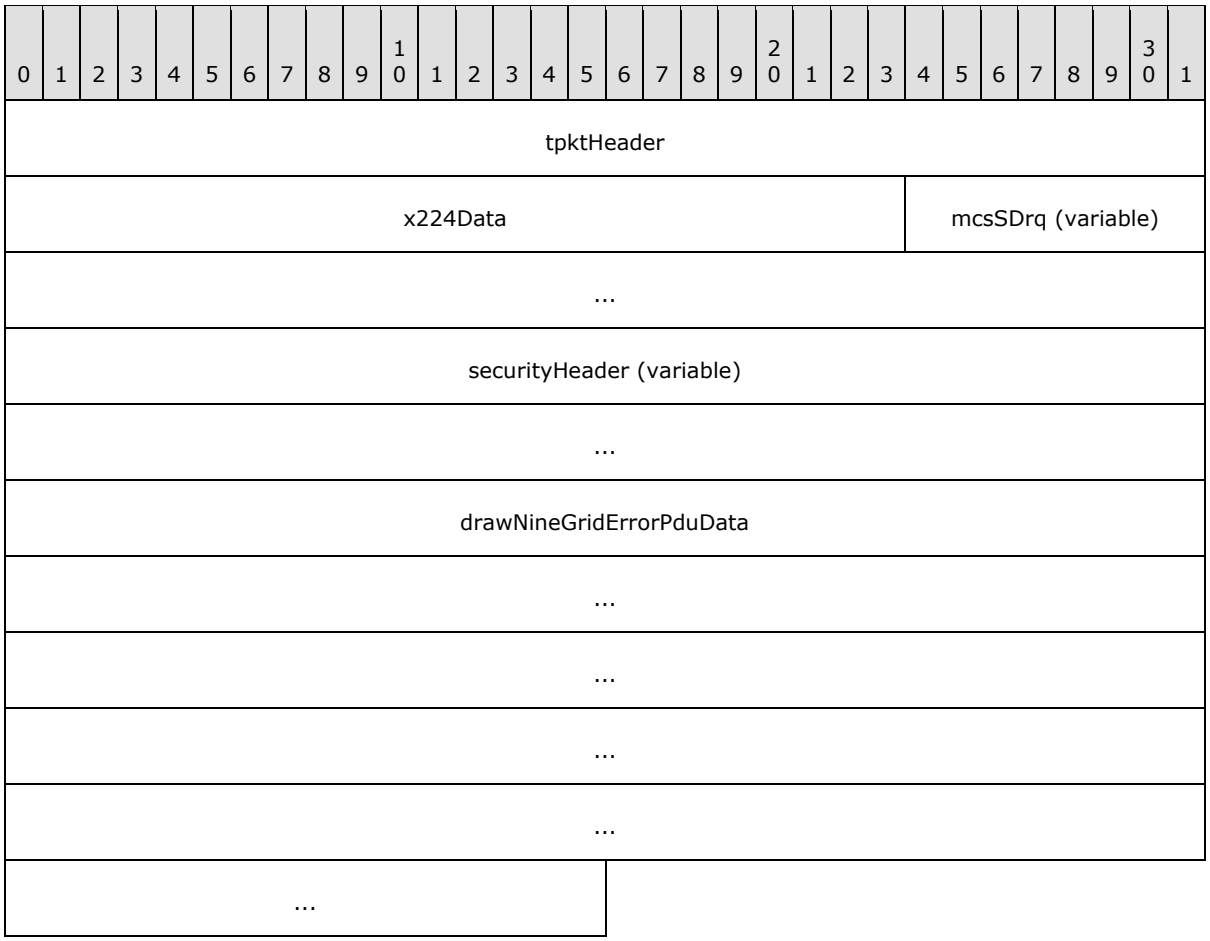
shareDataHeader (18 bytes): A Share Data Header (as specified in [MS-RDPBCGR] section 2.2.8.1.1.2.2) containing information on the packet. The type subfield of the **pduType** field of the Share Control Header within the Share Data Header MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_OFFSCRCACHE_ERROR_PDU (46).

flags (4 bytes): A 32-bit unsigned integer. Indicates the support for offscreen bitmap caching. This field MUST be set to **OC_ERR_FLUSH_AND_DISABLE_OFFSCREEN** (0x00000001), which specifies that the offscreen cache MUST be flushed and that further offscreen bitmap caching MUST be disabled.

Name	Value
OC_ERR_FLUSH_AND_DISABLE_OFFSCREEN	0x00000001

2.2.2.4.3 Client DrawNineGrid Cache Error PDU

The DrawNineGrid Cache Error PDU is sent by clients that support NineGrid bitmap caching and encounter caching errors.



tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable-length MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is specified in [T125] part 7 section 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the DrawNineGrid Cache Error PDU data.

securityHeader (variable): An optional security header. If Standard RDP Security is in effect, and the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), this field will contain one of the following headers:

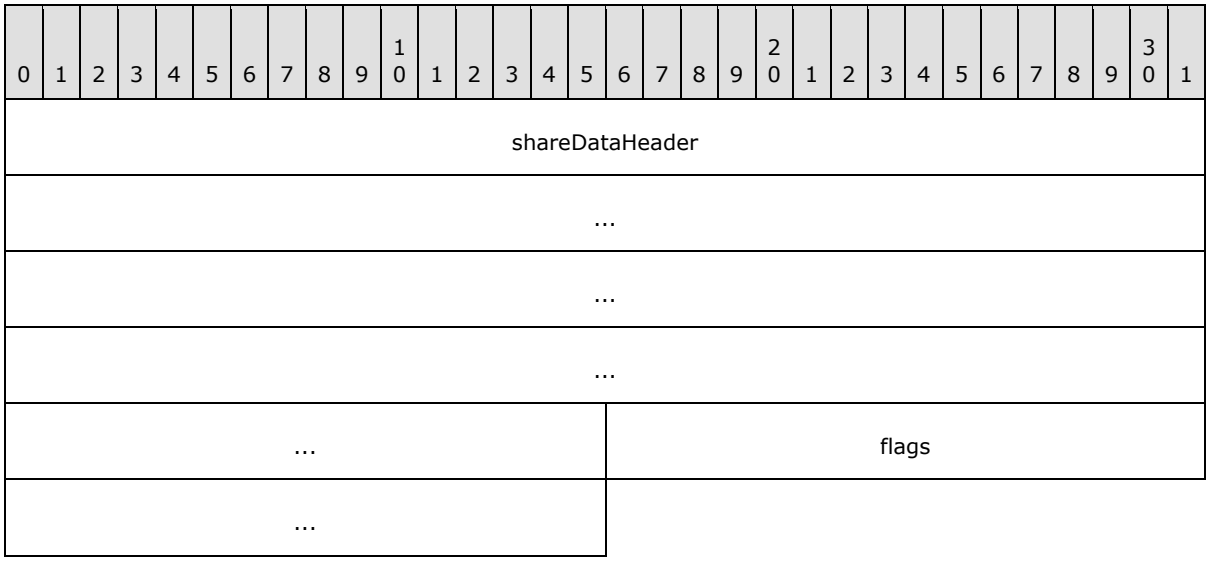
- Non-FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2), or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is in effect, or the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), this header is not included in the PDU.

drawNineGridErrorPduData (22 bytes): The actual contents of the DrawNineGrid Cache Error PDU, as specified in section 2.2.2.4.3.1.

2.2.2.4.3.1 DrawNineGrid Cache Error PDU Data (TS_DRAWNINEGRID_ERROR_PDU)

The TS_DRAWNINEGRID_ERROR_PDU structure contains the contents of the DrawNineGrid Cache Error PDU, which is essentially a Share Data Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) and a flags field.



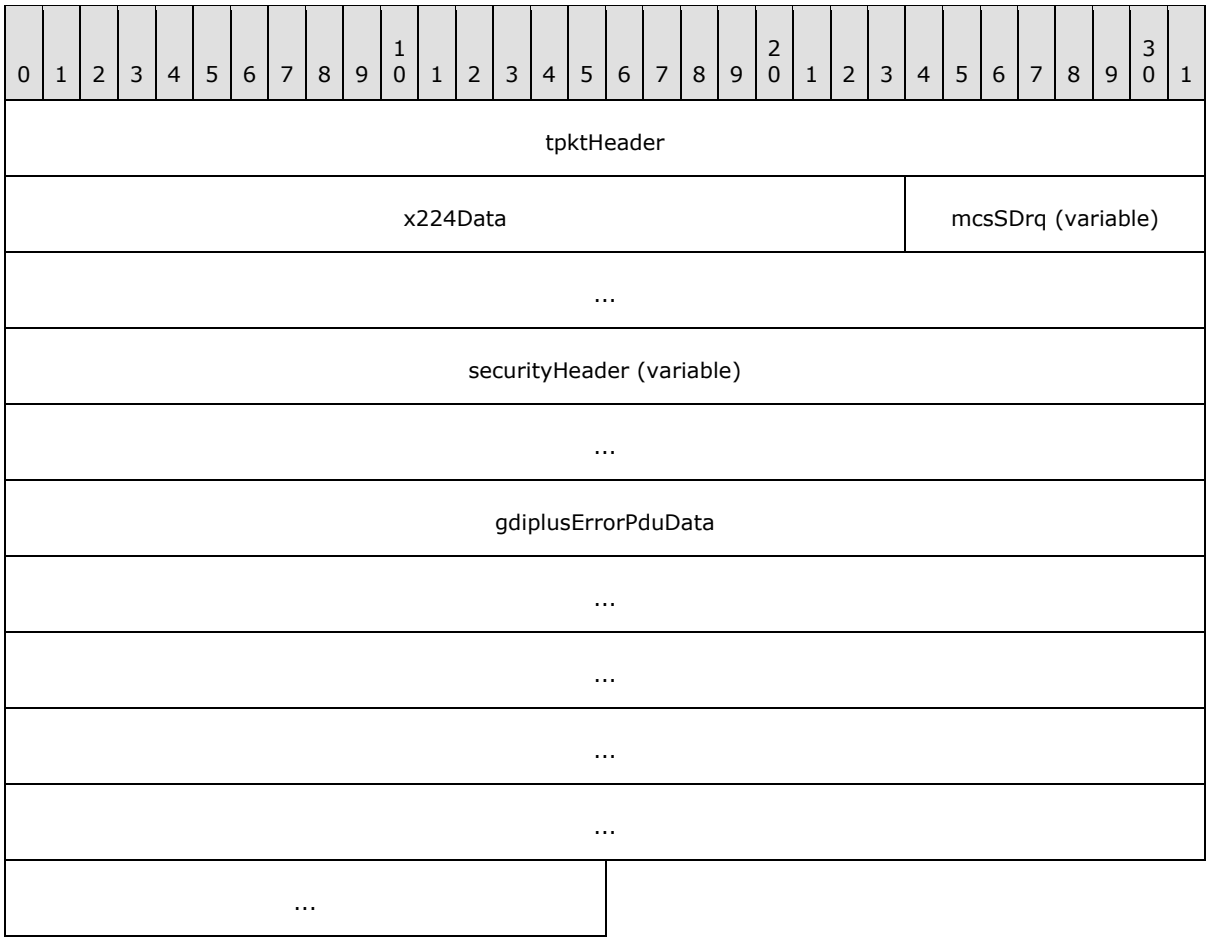
shareDataHeader (18 bytes): A Share Data Header (as specified in [MS-RDPBCGR] section 2.2.8.1.1.2.2) containing information on the packet. The type subfield of the **pduType** field of the Share Control Header within the Share Data Header MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_DRAWNINEGRID_ERROR_PDU (48).

flags (4 bytes): A 32-bit unsigned integer. Indicates support for NineGrid bitmap caching. This field MUST be set to DNG_ERR_FLUSH_AND_DISABLE_DRAWNINEGRID (0x00000001), which means that the NineGrid bitmap cache MUST be flushed and that further NineGrid bitmap caching MUST be disabled.

Name	Value
DNG_ERR_FLUSH_AND_DISABLE_DRAWNINEGRID	0x00000001

2.2.2.4.4 Client GDI+ Error PDU

The GDI+ Error PDU is sent by clients that support GDI+ 1.1 rendering and encounter errors.



tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDrq (variable): Variable-length MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is specified in [T125] part 7 section 7). The **userData** field of the MCS Send Data Request PDU contains a Security Header and the GDI+ Error PDU data.

securityHeader (variable): An optional security header. If Standard RDP Security is in effect, and the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), this field will contain one of the following headers:

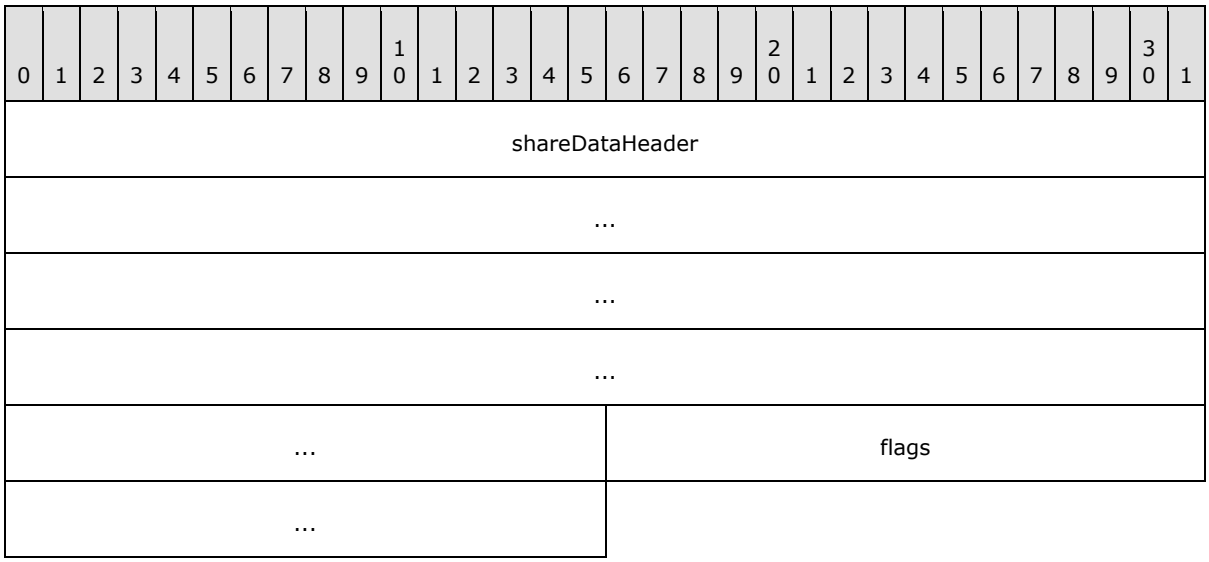
- Non-FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2), or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

If Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is in effect, or the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), this header is not included in the PDU.

gdiplusErrorPduData (22 bytes): The actual contents of the GDI+ Error PDU, as specified in section 2.2.2.4.4.1.

2.2.2.4.4.1 GDI+ Cache Error PDU Data (TS_DRAWGDIPLUS_ERROR_PDU)

The TS_DRAWGDIPLUS_ERROR_PDU structure contains the contents of the GDI+ Error PDU, which is essentially a Share Data Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) and a flags field.



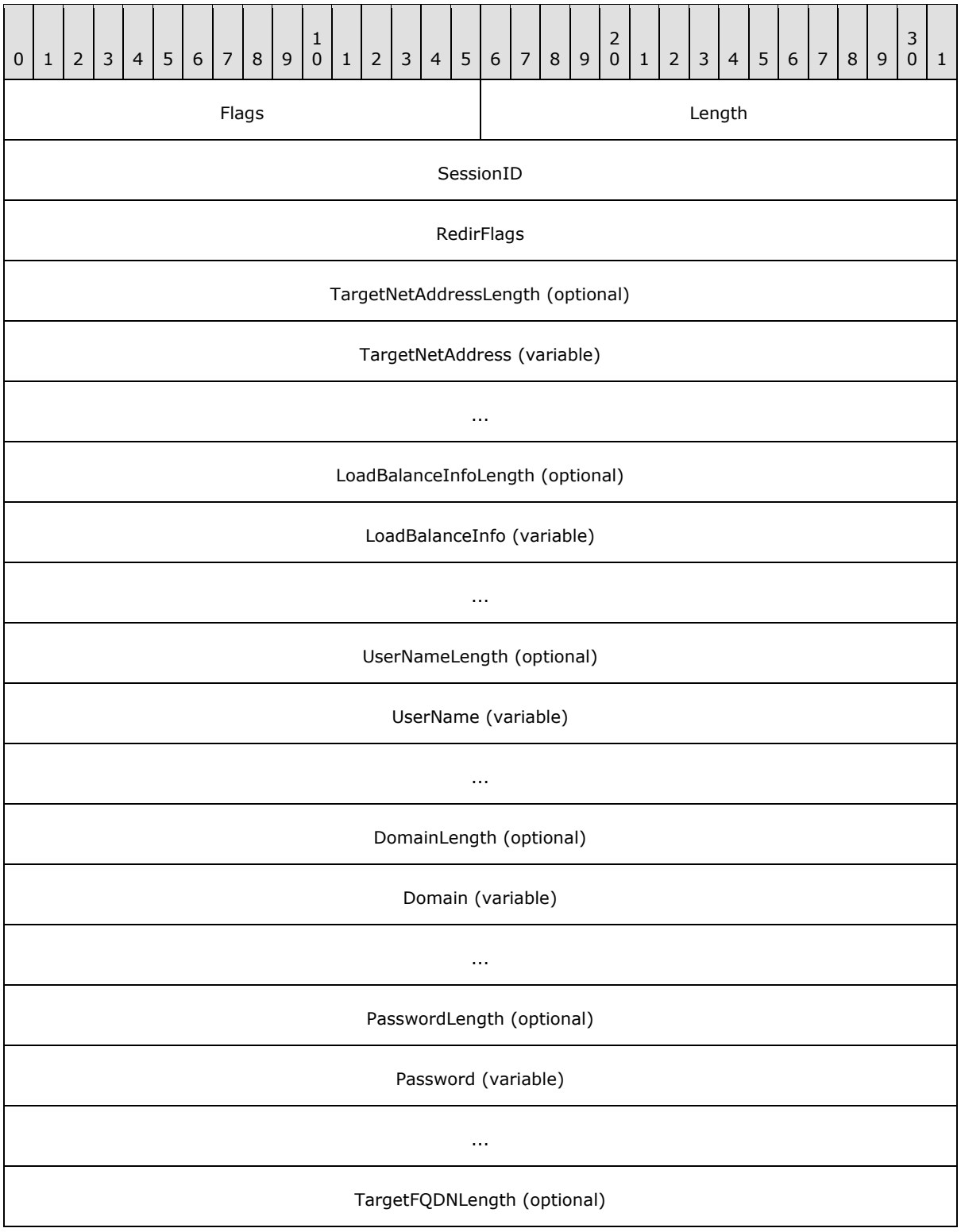
shareDataHeader (18 bytes): A Share Data Header containing information on the packet (see [MS-RDPBCGR] section 2.2.8.1.1.2.2). The type subfield of the **pduType** field of the Share Control Header within the Share Data Header MUST be set to PDUTYPE_DATAPDU (7). The **pduType2** field of the Share Data Header MUST be set to PDUTYPE2_DRAWGDIPLUS_ERROR_PDU (49).

flags (4 bytes): A 32-bit unsigned integer. Indicates support for GDI+ 1.1 rendering. This field MUST be set to GDIPLUS_ERR_FLUSH_AND_DISABLE_DRAWGDIPLUS (0x00000001), which means that the GDI+ drawing MUST be flushed and further GDI+ 1.1 rendering primitives MUST be disabled (implying that future drawings will be sent as bitmaps).

2.2.3 Server Redirection

2.2.3.1 Server Redirection Packet (RDP_SERVER_REDIRECTION_PACKET)

The RDP_SERVER_REDIRECTION_PACKET structure contains information to enable a client to reconnect to a session on a specified server. This data is sent to a client in a Redirection PDU to enable load-balancing of Terminal Server sessions across a collection of machines (for more information, see [MSFT-SDLBTS]).



TargetFQDN (variable)
...
TargetNetBiosNameLength (optional)
TargetNetBiosName (variable)
...
TargetNetAddressesLength (optional)
TargetNetAddresses (variable)
...

Flags (2 bytes): A 16-bit unsigned integer. The server redirection identifier. This field MUST be set to SEC_REDIRECTION_PKT (0x0400).

Length (2 bytes): A 16-bit unsigned integer. The overall length in bytes of the Server Redirection Packet structure.

SessionID (4 bytes): A 32-bit unsigned integer. The session identifier to which the client should reconnect. This identifier should be specified in the **RedirectSessionID** field of the Client Cluster Data (see [MS-RDPBCGR] section 2.2.1.3.5) if a reconnect attempt takes place. The Client Cluster Data is transmitted as part of the MCS Connect Initial PDU with GCC Conference Create Request (see [MS-RDPBCGR] section 2.2.1.3).

RedirFlags (4 bytes): A 32-bit unsigned integer. A bit field that contains redirection information flags, some of which indicate the presence of additional data at the end of the packet.

Value	Meaning
LB_TARGET_NET_ADDRESS 0x00000001	Indicates that the **TargetNetAddressLength** and **TargetNetAddress** fields are present.
LB_LOAD_BALANCE_INFO 0x00000002	Indicates that the **LoadBalanceInfoLength** and **LoadBalanceInfo** fields are present.
LB_USERNAME 0x00000004	Indicates that the **UserNameLength** and **UserName** fields are present.
LB_DOMAIN 0x00000008	Indicates that the **DomainLength** and **Domain** fields are present.
LB_PASSWORD 0x00000010	Indicates that the **PasswordLength** and **Password** fields are present.
LB_DONTSTOREUSERNAME 0x00000020	The client MUST not store the username in any internal data structures.

Value	Meaning
LB_SMARTCARD_LOGON 0x00000040	Indicates that the user can use a smartcard for authentication.
LB_NOREDIRECT 0x00000080	Indicates that the contents of the PDU are for informational purposes only. No actual redirection is required.
LB_TARGET_FQDN 0x00000100	Indicates that the **TargetFQDNLength** and **TargetFQDN** fields are present.
LB_TARGET_NETBIOS_NAME 0x00000200	Indicates that the **TargetNetBiosNameLength** and **TargetNetBiosName** fields are present.
LB_TARGET_NET_ADDRESSES 0x00000800	Indicates that the **TargetNetAddressesLength** and **TargetNetAddresses** fields are present.

TargetNetAddressLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **TargetNetAddress** field.

TargetNetAddress (variable): An array of bytes containing the IP address of the server (for example, "192.168.0.1" using dotted decimal notation) in Unicode format, including a null-terminator.

LoadBalanceInfoLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **LoadBalanceInfo** field.

LoadBalanceInfo (variable): An array of bytes containing load balancing information that MUST be treated as opaque data by the client and passed to the server (if a reconnection takes place) in the **routingToken** field of the X.224 Connection Request PDU (see [MS-RDPBCGR] section 2.2.1.1).

UserNameLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **UserName** field.

UserName (variable): An array of bytes containing the username of the user in Unicode format, including a null-terminator.

DomainLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **Domain** field.

Domain (variable): An array of bytes containing the domain to which the user connected in Unicode format, including a null-terminator.

PasswordLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **Password** field.

Password (variable): An array of bytes containing the password used by the user in Unicode format, including a null-terminator or a cookie value that MUST be passed to the target server on successful connection.

TargetFQDNLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **TargetFQDN** field.

TargetFQDN (variable): An array of bytes containing the fully-qualified domain name (FQDN) of the target machine, including a null-terminator.

TargetNetBiosNameLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **TargetNetBiosName** field.

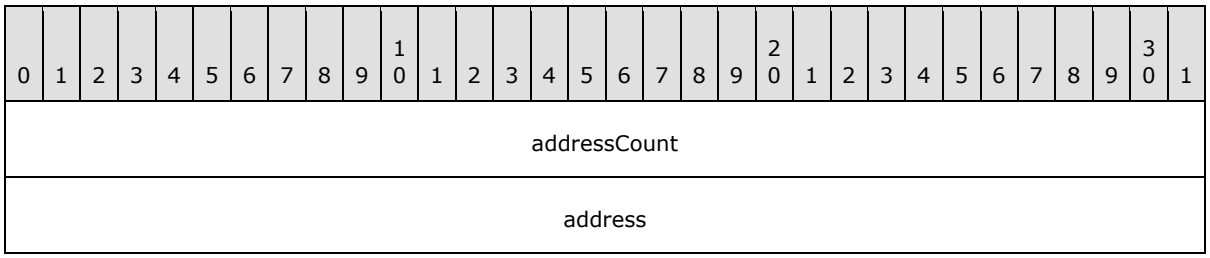
TargetNetBiosName (variable): An array of bytes containing the NETBIOS name of the target machine, including a null-terminator.

TargetNetAddressesLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the **TargetNetAddresses** field.

TargetNetAddresses (variable): An array of bytes containing the target IP addresses of the server to connect against, stored in a Target Net Addresses structure (see section 2.2.3.1.1).

2.2.3.1.1 Target Net Addresses (TARGET_NET_ADDRESSES)

The TARGET_NET_ADDRESSES structure is used to hold a collection of IP addresses in Unicode format.

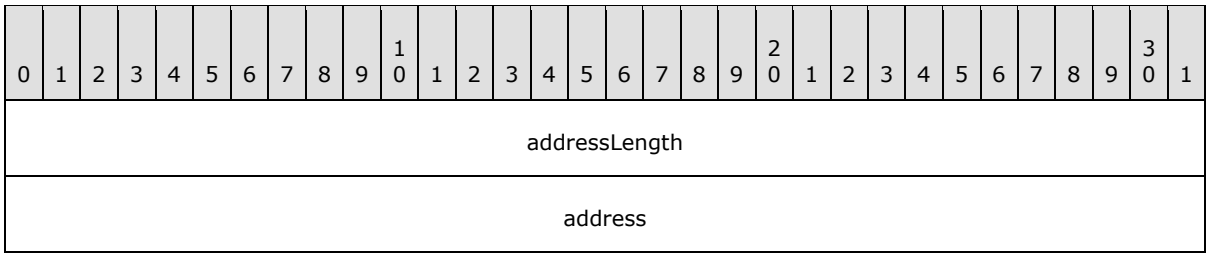


addressCount (4 bytes): A 32-bit unsigned integer. The number of IP addresses present in the address field.

address (4 bytes): An array of Target Net Address (section 2.2.3.1.1.1) structures, each containing an IP address.

2.2.3.1.1.1 Target Net Address (TARGET_NET_ADDRESS)

The Target Net Address structure holds a Unicode text representation of an IP address.



addressLength (4 bytes): A 32-bit unsigned integer. The length in bytes of the address field.

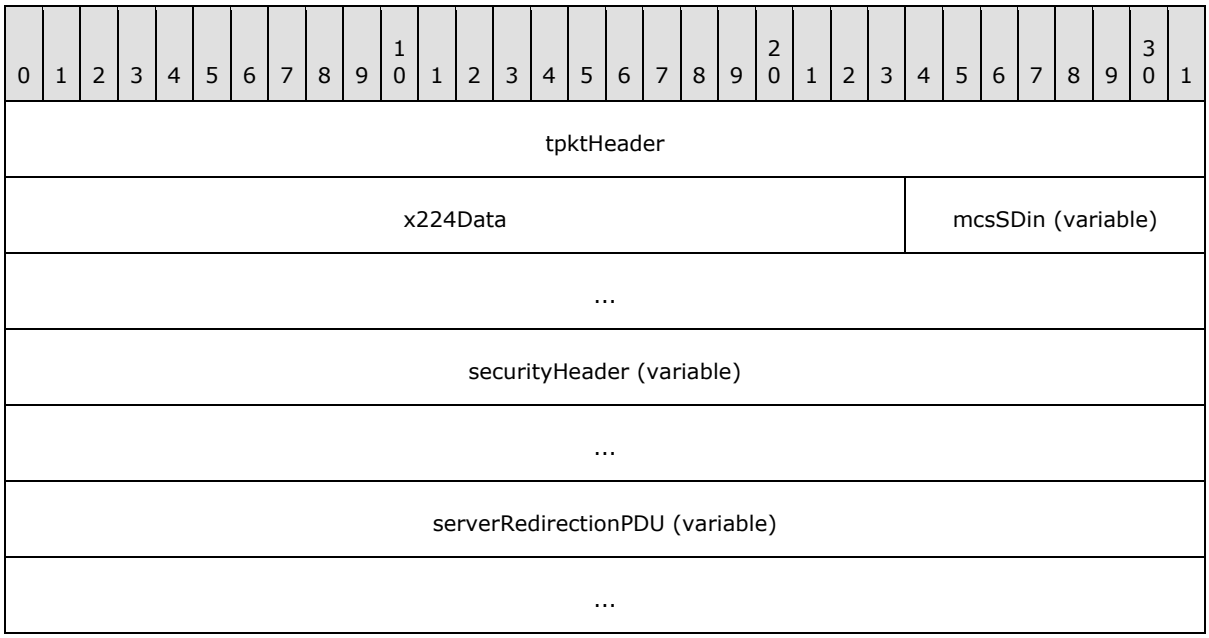
address (4 bytes): An array of bytes containing an IP address in Unicode format, including a null-terminator.

2.2.3.2 Standard RDP Security

2.2.3.2.1 Standard Security Server Redirection PDU

The Standard Security Server Redirection PDU is sent by the server to the client to instruct it to reconnect to an existing session on another server. The information required to perform the reconnection is contained in an embedded Server Redirection Packet (see section 2.2.3.1). This PDU

MUST NOT be sent if Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is in effect. The Enhanced Security Server Redirection PDU (see section 2.2.3.3.1) MUST be used instead.



tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable-length MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is specified in [T125] part 7 section 7). The userData field of the MCS Send Data Indication PDU contains a Security Header and the Server Redirection PDU data.

securityHeader (variable): An optional security header. If Standard RDP Security is in effect, and the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is greater than ENCRYPTION_METHOD_NONE (0), this field will contain one of the following headers:

- Non-FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.2) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_LOW (1), ENCRYPTION_LEVEL_CLIENT_COMPATIBLE (2), or ENCRYPTION_LEVEL_HIGH (3).
- FIPS Security Header (see [MS-RDPBCGR] section 2.2.8.1.1.2.3) if the Encryption Level selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_LEVEL_FIPS (4).

The **flags** field of the security header MUST contain the SEC_REDIRECTION_PKT (0x0400) flag (see [MS-RDPBCGR] section 2.2.8.1.1.2.1).

If the Encryption Method selected by the server (see [MS-RDPBCGR] sections 5.3.2 and 2.2.1.4.3) is ENCRYPTION_METHOD_NONE (0), this header is not included in the PDU.

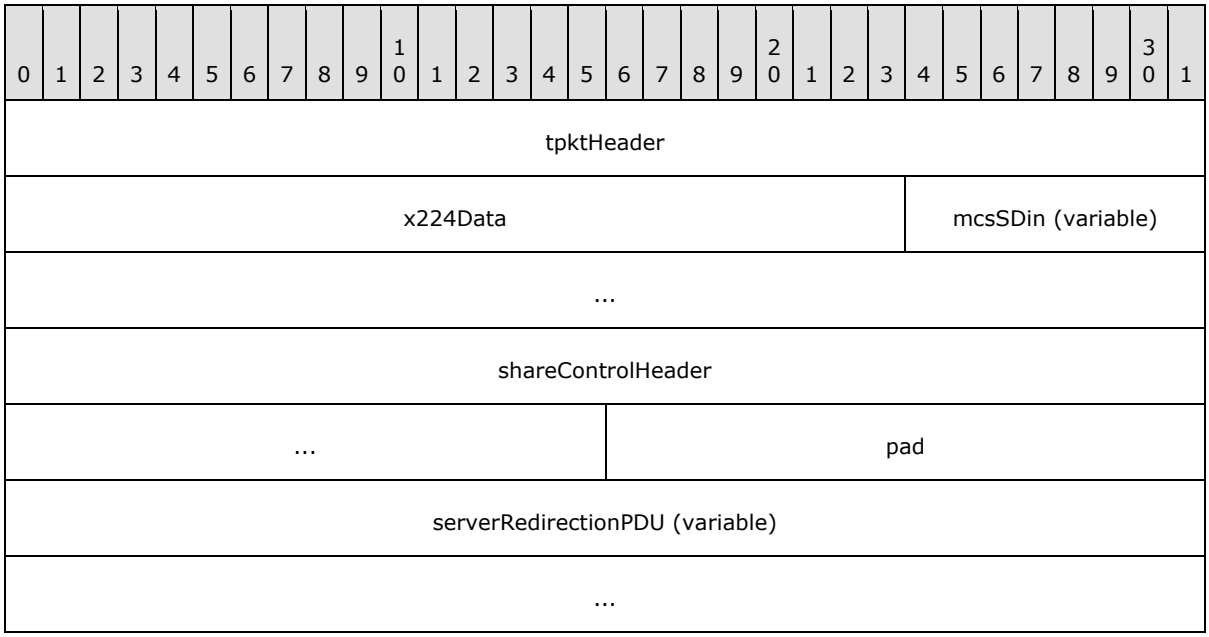
This PDU MUST NOT be sent if Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is in effect. The Enhanced Security Server Redirection PDU (see section 2.2.3.3.1) MUST be used instead.

serverRedirectionPDU (variable): Information required by the client to initiate a reconnection to a given session on a target server encapsulated in a Server Redirection Packet (section 2.2.3.1) structure.

2.2.3.3 Enhanced RDP Security

2.2.3.3.1 Enhanced Security Server Redirection PDU

The Enhanced Security Server Redirection PDU is sent by the server to the client to instruct it to reconnect to an existing session on another server. The information required to perform the reconnection is contained in an embedded Server Redirection Packet (section 2.2.3.1). This PDU MUST NOT be sent if Standard RDP Security (see [MS-RDPBCGR] section 5.3) is in effect. The Standard Security Server Redirection PDU (see section 2.2.3.2.1) MUST be used instead.



tpktHeader (4 bytes): A TPKT Header, as specified in [T123] section 8.

x224Data (3 bytes): An X.224 Class 0 Data TPDU, as specified in [X224] section 13.7.

mcsSDin (variable): Variable-length MCS Send Data Request PDU, as specified in [T125] (the ASN.1 structure definition is specified in [T125] part 7 section 7). The userData field of the MCS Send Data Indication PDU contains a Security Header and the Server Redirection PDU data.

shareControlHeader (6 bytes): A Share Control Header (as specified in [MS-RDPBCGR] section 2.2.8.1.1.1.1) containing information on the packet. The type subfield of the **pduType** field of the Share Control Header MUST be set to PDUTYPE_SERVER_REDIR_PKT (10). The versionHigh and versionLow subfields MUST both be set to 0.

pad (2 bytes): A 16-bit unsigned integer. Padding. Values in this field are arbitrary, and MUST be ignored.

serverRedirectionPDU (variable): Information required by the client to initiate a reconnection to a given session on a target server encapsulated in a Server Redirection Packet (section 2.2.3.1) structure.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

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

3.1.1.1 Caches

All of the RDP caches are managed by the server. The server decides what items to cache and in what cache entries these items MUST be stored. These items, and the metadata describing the cache and cache entry, are sent to the client using an appropriate drawing order. The layout of most of the caches is usually negotiated using a suitable capability set.

3.1.1.1.1 Bitmap Caches

Bitmap caches are used by the client and server to store graphic bitmaps. Each bitmap cache holds bitmaps of a specified size in pixels (known as the "tile size"). If a bitmap does not fit into a single cache entry, the server uses a tiling algorithm to divide the bitmap into tiles that will fit into the cache entries so that they can be stored separately into the cache.

The Cache Bitmap - Revision 1 (section 2.2.2.3.1.2.2) and Cache Bitmap - Revision 2 (section 2.2.2.3.1.2.3) Secondary Drawing Orders are used to update the contents of the client-side bitmap cache.

There are two versions of bitmap caches. Revision 1 supports only memory-based caching. Revision 2 supports persistent disk caching (in addition to memory caching) by sending a 64-bit key (derived from a cryptographic hash of the bitmap contents) for each bitmap. These 64-bit keys can be sent to the server on subsequent connections (using the Persistent Key List PDU specified in [MS-RDPBCGR] section 2.2.1.17.1) to initialize the persistent disk cache. The server maintains its own cache of bitmap keys that it has already sent to the client (see section 3.3.1.1).

When using Revision 2 bitmap caching, the client can request that the server delays forcing it to cache a bitmap to disk until the bitmap has been used more than once. This is implemented by requesting that the server maintains a Bitmap Cache Wait List (see section 3.3.1.3). Once a particular bitmap has been encountered by the server more than once, it will then instruct the client to cache it (see section 2.2.2.3.1.2.3).

Support for Revision 2 caching is advertised by the server by using the Bitmap Cache Host Support Capability Set (see [MS-RDPBCGR] section 2.2.7.1.4). Client support for Revision 1 bitmap caching is advertised by using the Revision 1 Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.1) while support for Revision 2 bitmap caching is advertised by using the Revision 2 Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.2).

The Revision 1 cache defaults to: 1.5 MB of memory cache for 8 bpp, 3 MB for 16 bpp, 4.5 MB for 24 bpp, and 6 MB for 32 bpp. Revision 2 defaults to: 10 MB of persistent cache storage for 8 bpp, 20 MB for 16 bpp, 30 MB for 24 bpp, and 40 MB for 32 bpp. The actual layout for the bitmap caches are negotiated in the Bitmap Cache Capability Sets (see [MS-RDPBCGR] section 2.2.7.1.5).

The default Revision 1 Bitmap Cache layout follows:

Bitmap cache ID	Tile size (in pixels)	Cache entry size (in bytes)	Number of cache entries (default / maximum)
0	16 x 16	256	120 / 600
1	32 x 32	1024	120 / 600
2	64 x 64	4096	Rest of 1.5 MB

The default Revision 2 Bitmap Cache layout follows:

Bitmap cache ID	Tile size (in pixels)	Cache entry size (in bytes)	Number of cache entries
0	16 x 16	256	120
1	32 x 32	1024	120
2	64 x 64	4096	Depends on disk cache

If persistent bitmap caching is enabled, the client MUST enumerate the entries in its local persistent bitmap cache to obtain the 64-bit bitmap keys for all of the stored bitmaps, and then send one or more Persistent Key List PDUs to the server that contain this data (see [MS-RDPBCGR] section 2.2.1.17).

Bitmap images stored in the bitmap caches are rendered using the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders.

3.1.1.1.2 Glyph and Fragment Caches

Glyph caching supports 10 glyph caches and 1 fragment cache to store bitmaps of font characters in memory. Glyphs are first cached in the glyph caches before being displayed. A fragment is a set of glyphs defined in terms of glyph indices.

The Cache Glyph - Revision 1 (section 2.2.2.3.1.2.5) and Cache Glyph - Revision 2 (section 2.2.2.3.1.2.6) Secondary Drawing Orders are used to update the glyph caches. The GlyphIndex (section 2.2.2.3.1.1.2.13), FastIndex (section 2.2.2.3.1.1.2.14), and FastGlyph (section 2.2.2.3.1.1.2.15) Primary Drawing Orders consume glyphs from the glyph caches. (The GlyphIndex and FastGlyph Orders also populate the Fragment Cache.)

The actual layout of the glyph and fragment caches are negotiated in the Glyph Cache Capability Sets (see [MS-RDPBCGR] section 2.2.7.1.9).

The default Glyph Cache layout follows:

Glyph cache ID	Cache entry size (in bytes)	Number of cache entries
0	4	254
1	4	254
2	8	254
3	8	254
4	16	254

Glyph cache ID	Cache entry size (in bytes)	Number of cache entries
5	32	254
6	64	254
7	128	254
8	256	254
9	256	64

There is only 1 fragment cache, and by default it has 256 entries with 256 bytes as the entry size.

3.1.1.1.3 Color Table Cache

The Color Table Cache is used to reduce bandwidth by caching color palettes. The existence of the Color Table Cache is implicitly tied to support for the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders. If support for these orders is advertised in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3), the existence of a color table cache with entries for six palettes is implied when palletized color is being used. Each of the 6 cached color tables holds 256 color mappings, initialized by the Cache Color Table (section 2.2.2.3.1.2.4) Secondary Drawing Order. Cached color tables are used exclusively by the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders.

3.1.1.1.4 Brush Caches

There are two brush caches, a mono brush cache and a color brush cache. Each cache can hold 64 brush entries. The size of a mono brush is 16 bytes while the size of a color brush is 64 bytes for 8 bpp and 192 bytes for 24 bpp. The brush cache sizes are not negotiated. However, support for brush caching is negotiated using the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8). The Cache Brush (section 2.2.2.3.1.2.7) Secondary Drawing Order is used to populate the brush caches.

3.1.1.1.5 Offscreen Bitmap Cache

The Offscreen Bitmap Cache is used to store writable offscreen bitmap surfaces. There is only one cache for all offscreen bitmaps. The bitmap sizes are variable and depend on the dimensions specified by the Create Offscreen Bitmap (section 2.2.2.3.1.3.2) Alternate Secondary Drawing Order.

The total cache size and cache entries are capped by the values negotiated in the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10). The default size of the cache is 2.5 MB for 8 bpp, 5 MB for 16 bpp, or 7.5 MB for 24 bpp. The default number of allowed cache entries is 100.

The Create Offscreen Bitmap (section 2.2.2.3.1.3.2) and Switch Surface (section 2.2.2.3.1.3.3) Alternate Secondary Drawing Orders are used to manipulate the offscreen cache. Images stored in the Offscreen Bitmap Cache are rendered using the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders.

3.1.1.1.6 NineGrid Bitmap Cache

The NineGrid bitmap cache is used to store NineGrid-compliant bitmaps. There is only one cache for all NineGrid bitmaps. The individual bitmap sizes are variable and depend on the bitmap dimensions.

The total cache size and cache entries are capped by the DrawNineGrid Cache Capability Set (section 2.2.1.2). The default size of the cache is 2.5 MB for 8 bpp, 5 MB for 16 bpp, or 7.5 MB for 24 bpp. The default number of allowed cache entries is 256.

The Create NineGrid Bitmap (section 2.2.2.3.1.3.4), Stream Bitmap First (section 2.2.2.3.1.3.5.1), and Stream Bitmap Next (section 2.2.2.3.1.3.5.2) Alternate Secondary Drawing Orders are used to populate the NineGrid Bitmap Cache. Individual bitmaps within the cache are rendered with the DrawNineGrid (section 2.2.2.3.1.1.2.21) Primary Drawing Order. This primary drawing order applies a NineGrid algorithm to the bitmap before rendering it (for an example rendering, see section 4.6).

3.1.1.1.7 GDI+ Caches

The GDI+ caches are used to store Graphics, Brush, Pen, Image, and Image Attributes primitives. The total cache sizes and cache entries permitted are negotiated using the Draw GDI+ Capability Set (section 2.2.1.3).

3.1.2 Timers

No common timers are used.

3.1.3 Initialization

No initialization steps are specified.

3.1.4 Higher-Layer Triggered Events

No common higher-layer triggered events are used.

3.1.5 Message Processing Events and Sequencing Rules

There are no common message processing events or sequencing rules. Specification of message processing events and rules is deferred to sections 3.2.5 and 3.3.5.

3.1.6 Timer Events

No common timer events are used.

3.1.7 Other Local Events

No common additional events are used.

3.1.8 RDP 6.0-Based Bulk Data Compression

RDP version 6.0 supports an extension to the MPPC-based bulk compressor described in [MS-RDPBCGR] section 3.1.8. This extension is called "RDP 6.0 Bulk Compression" (RDP6-BC), and is only supported for server-to-client traffic.

3.1.8.1 Abstract Data Model

The shared state necessary to support the transmission and reception of RDP6-BC compressed data between a client and server requires a history buffer and a current offset into the history buffer (HistoryOffset). The size of the history buffer is 64 KB. The history buffer and HistoryOffset MUST both start initialized to zero.

In addition to the history buffer and HistoryOffset, a small cache MUST also be managed by the client and server endpoints. This cache is referred to as the OffsetCache and is used to store the last four unique copy-offsets encountered during data compression (copy-offsets are described in [MS-RDPBCGR] section 3.1.8.1). This saves on bandwidth in cases in which there are many repeated copy-offsets.

While compressing data, the sender endpoint inserts the uncompressed data at the position in the history buffer given by the HistoryOffset. After insertion, the HistoryOffset is advanced by the amount of data added. If the data does not fit into the history buffer (the sum of the HistoryOffset and the size of the uncompressed data exceeds the size of the history buffer), the history buffer MUST be slid back by half its size, and the HistoryOffset MUST be reset to the middle of the history buffer (offset 32768).

As the receiver endpoint decompresses the data, it inserts the decompressed data at the position in the history buffer given by its local copy HistoryOffset. If a slide-back occurs, the sender endpoint MUST notify the target receiver so it can reset its local state. In this way, the sender and receiver endpoints maintain an exact replica of the history buffer and HistoryOffset.

3.1.8.2 Compressing Data

Compression using RDP6-BC is based on the principles outlined in [MS-RDPBCGR] section 3.1.8.2 with literals and copy-tuples being encoded using the scheme described in section 3.1.8.4.3. Also, due to the slide-back behavior outlined in section 3.1.8.1, the meaning of the PACKET_AT_FRONT flag (0x40) has changed. This flag MUST still be set in conjunction with the PACKET_COMPRESSED (0x80) flag; however, it is now used to indicate that the history buffer MUST be slid back by half its size and that the HistoryOffset MUST be reset to the middle of the history buffer.

3.1.8.3 Decompressing Data

Decompression using RDP6-BC is based on the principles specified in [MS-RDPBCGR] section 3.1.8.3 with the compressed stream being decoded using the scheme described in section 3.1.8.4.4. Also, due to the slide-back behavior described in section 3.1.8.1, the interpretation of the PACKET_AT_FRONT flag (0x40) has changed. If this flag is set, the decompressor MUST slide the history buffer back by half its size and reset the HistoryOffset to the middle of the history buffer.

3.1.8.4 Wire Format

The compressed stream consists of a bit-encoded sequence of literals, <copy-offset, length-of-match> tuples (also known as copy-tuples), and a terminating End-of-Stream (EOS) marker.

Literals, copy-offsets, and EOS marker values are Huffman encoded to produce 294 individual LiteralOrEosOrCopyOffset variable-length bit-codes (possibly followed by extra bits of information). Each LiteralOrEosOrCopyOffset code is optionally followed by 1 of 32 possible Huffman encoded length-of-match codes (LengthOfMatch) that is present, depending on the contents of the previous LiteralOrEosOrCopyOffset code (the LengthOfMatch code is also possibly followed by extra bits of information).

The convention used on the stream is that the sender MUST output bytes in little-endian order with the most significant bit first.

For more information on Huffman encoding, see [HUFFCODE].

3.1.8.4.1 Literal, EOS, and Copy-Offset Tables

The length in bits for each of the 0 to 293 Huffman-encoded LiteralOrEosOrCopyOffset codes are listed in table 1 (HuffLengthLEC):

Code index	0	1	2	3	4	5	6	7
0	0x6	0x6	0x6	0x7	0x7	0x7	0x7	0x7
8	0x7	0x7	0x7	0x8	0x8	0x8	0x8	0x8
16	0x8	0x8	0x9	0x8	0x9	0x9	0x9	0x9
24	0x8	0x8	0x9	0x9	0x9	0x9	0x9	0x9
32	0x8	0x9	0x9	0xa	0x9	0x9	0x9	0x9
40	0x9	0x9	0x9	0xa	0x9	0xa	0xa	0xa
48	0x9	0x9	0xa	0x9	0xa	0x9	0xa	0x9
56	0x9	0x9	0xa	0xa	0x9	0xa	0x9	0x9
64	0x8	0x9	0x9	0x9	0x9	0xa	0xa	0xa
72	0x9	0x9	0xa	0xa	0xa	0xa	0xa	0xa
80	0x9	0x9	0xa	0xa	0xa	0xa	0xa	0xa
88	0xa	0x9	0xa	0xa	0xa	0xa	0xa	0xa
96	0x8	0xa	0xa	0xa	0xa	0xa	0xa	0xa
104	0xa	0xa	0xa	0xa	0xa	0xa	0xa	0xa
112	0x9	0xa	0xa	0xa	0xa	0xa	0xa	0xa
120	0x9	0xa	0xa	0xa	0xa	0xa	0xa	0x9
128	0x7	0x9	0x9	0xa	0x9	0xa	0xa	0xa
136	0x9	0xa	0xa	0xa	0xa	0xa	0xa	0xa
144	0x9	0xa	0xa	0xa	0xa	0xa	0xa	0xa
152	0xa	0xa	0xa	0xa	0xa	0xa	0xa	0xa
160	0xa	0xa	0xa	0xa	0xa	0xa	0xa	0xa
168	0xa	0xa	0xa	0xd	0xa	0xa	0xa	0xa
176	0xa	0xa	0xb	0xa	0xa	0xa	0xa	0xa
184	0xa	0xa	0xa	0xa	0xa	0xa	0xa	0xa
192	0x9	0xa	0xa	0xa	0xa	0xa	0x9	0xa
200	0xa	0xa	0xa	0xa	0x9	0xa	0xa	0xa
208	0x9	0xa	0xa	0xa	0xa	0xa	0xa	0xa

Code index	0	1	2	3	4	5	6	7
216	0xa	0xa	0xa	0xa	0xa	0xa	0xa	0xa
224	0x9	0xa	0xa	0xa	0xa	0xa	0xa	0xa
232	0xa	0xa	0xa	0xa	0xa	0xa	0x9	0xa
240	0x8	0x9	0x9	0xa	0x9	0xa	0xa	0xa
248	0x9	0xa	0xa	0xa	0x9	0x9	0x8	0x7
256	0xd	0xd	0x7	0x7	0xa	0x7	0x7	0x6
264	0x6	0x6	0x6	0x5	0x6	0x6	0x6	0x5
272	0x6	0x5	0x6	0x6	0x6	0x6	0x6	0x6
280	0x6	0x6	0x6	0x6	0x6	0x6	0x6	0x6
288	0x8	0x5	0x6	0x7	0x7	0xD		

Table 1: Bit-lengths for the 294 Huffman-encoded LiteralOrEosOrCopyOffset codes

For example, from table 1, it can be determined that the 0th Huffman-encoded LiteralOrEosOrCopyOffset code has a length of 6 bits while the 131st Huffman-encoded LiteralOrEosOrCopyOffset code has a length of 10 (0x0A) bits.

Using the canonical Huffman algorithm (for more information, see [CANONHUFF]), the Huffman codebook table shown in table 2 (HuffCodeLEC) can be obtained. The bit-lengths in table 1 MUST be used to isolate the appropriate bits.

Code index	0	1	2	3	4	5	6	7
0	0x0004	0x0024	0x0014	0x0011	0x0051	0x0031	0x0071	0x0009
8	0x0049	0x0029	0x0069	0x0015	0x0095	0x0055	0x00d5	0x0035
16	0x00b5	0x0075	0x001d	0x00f5	0x011d	0x009d	0x019d	0x005d
24	0x000d	0x008d	0x015d	0x00dd	0x01dd	0x003d	0x013d	0x00bd
32	0x004d	0x01bd	0x007d	0x006b	0x017d	0x00fd	0x01fd	0x0003
40	0x0103	0x0083	0x0183	0x026b	0x0043	0x016b	0x036b	0x00eb
48	0x0143	0x00c3	0x02eb	0x01c3	0x01eb	0x0023	0x03eb	0x0123
56	0x00a3	0x01a3	0x001b	0x021b	0x0063	0x011b	0x0163	0x00e3
64	0x00cd	0x01e3	0x0013	0x0113	0x0093	0x031b	0x009b	0x029b
72	0x0193	0x0053	0x019b	0x039b	0x005b	0x025b	0x015b	0x035b
80	0x0153	0x00d3	0x00db	0x02db	0x01db	0x03db	0x003b	0x023b
88	0x013b	0x01d3	0x033b	0x00bb	0x02bb	0x01bb	0x03bb	0x007b

Code index	0	1	2	3	4	5	6	7
96	0x002d	0x027b	0x017b	0x037b	0x00fb	0x02fb	0x01fb	0x03fb
104	0x0007	0x0207	0x0107	0x0307	0x0087	0x0287	0x0187	0x0387
112	0x0033	0x0047	0x0247	0x0147	0x0347	0x00c7	0x02c7	0x01c7
120	0x0133	0x03c7	0x0027	0x0227	0x0127	0x0327	0x00a7	0x00b3
128	0x0019	0x01b3	0x0073	0x02a7	0x0173	0x01a7	0x03a7	0x0067
136	0x00f3	0x0267	0x0167	0x0367	0x00e7	0x02e7	0x01e7	0x03e7
144	0x01f3	0x0017	0x0217	0x0117	0x0317	0x0097	0x0297	0x0197
152	0x0397	0x0057	0x0257	0x0157	0x0357	0x00d7	0x02d7	0x01d7
160	0x03d7	0x0037	0x0237	0x0137	0x0337	0x00b7	0x02b7	0x01b7
168	0x03b7	0x0077	0x0277	0x07ff	0x0177	0x0377	0x00f7	0x02f7
176	0x01f7	0x03f7	0x03ff	0x000f	0x020f	0x010f	0x030f	0x008f
184	0x028f	0x018f	0x038f	0x004f	0x024f	0x014f	0x034f	0x00cf
192	0x000b	0x02cf	0x01cf	0x03cf	0x002f	0x022f	0x010b	0x012f
200	0x032f	0x00af	0x02af	0x01af	0x008b	0x03af	0x006f	0x026f
208	0x018b	0x016f	0x036f	0x00ef	0x02ef	0x01ef	0x03ef	0x001f
216	0x021f	0x011f	0x031f	0x009f	0x029f	0x019f	0x039f	0x005f
224	0x004b	0x025f	0x015f	0x035f	0x00df	0x02df	0x01df	0x03df
232	0x003f	0x023f	0x013f	0x033f	0x00bf	0x02bf	0x014b	0x01bf
240	0x00ad	0x00cb	0x01cb	0x03bf	0x002b	0x007f	0x027f	0x017f
248	0x012b	0x037f	0x00ff	0x02ff	0x00ab	0x01ab	0x006d	0x0059
256	0x17ff	0x0fff	0x0039	0x0079	0x01ff	0x0005	0x0045	0x0034
264	0x000c	0x002c	0x001c	0x0000	0x003c	0x0002	0x0022	0x0010
272	0x0012	0x0008	0x0032	0x000a	0x002a	0x001a	0x003a	0x0006
280	0x0026	0x0016	0x0036	0x000e	0x002e	0x001e	0x003e	0x0001
288	0x00ed	0x0018	0x0021	0x0025	0x0065	0x1fff		

Table 2: Huffman codebook for the 294 Huffman-encoded LiteralOrEosOrCopyOffset codes

For example: From table 2, it can be determined that the 0th Huffman-encoded LiteralOrEosOrCopyOffset code has a value of 0x0004. Applying the bit length information from table 1 (6 bits), it can be determined that the Huffman code in binary MUST be 000100.

Consider another example: From table 2, it can be determined that the 131st Huffman-encoded LiteralOrEosOrCopyOffset code has a value of 0x02a7. Applying the bit length information from table 1 (10 bits), it can be determined that the Huffman code in binary MUST be 1010100111.

The lookup tables in table 3 are used during encoding and decoding of the copy-offset for a given copy-tuple to determine the extra bits that will follow the LiteralOrEosOrCopyOffset code.

Index	CopyOffsetBitsLUT	CopyOffsetBaseLUT
0	0	1
1	0	2
2	0	3
3	0	4
4	1	5
5	1	7
6	2	9
7	2	13
8	3	17
9	3	25
10	4	33
11	4	49
12	5	65
13	5	97
14	6	129
15	6	193
16	7	257
17	7	385
18	8	513
19	8	769
20	9	1025
21	9	1537
22	10	2049
23	10	3073
24	11	4097
25	11	6145

Index	CopyOffsetBitsLUT	CopyOffsetBaseLUT
26	12	8193
27	12	12289
28	13	16385
29	13	24577
30	14	32769
31	14	49153
32	15	65537

Table 3: Bit count and base value lookup tables to encode and decode copy-offset values

3.1.8.4.2 Length-of-Match Tables

The length in bits for each of the 0 to 31 Huffman-encoded LengthOfMatch codes are listed in table 4 (HuffLengthL):

	0	1	2	3	4	5	6	7
0	0x4	0x2	0x3	0x4	0x3	0x4	0x4	0x5
8	0x4	0x5	0x5	0x6	0x6	0x7	0x7	0x8
16	0x7	0x8	0x8	0x9	0x9	0x8	0x9	0x9
24	0x9	0x9	0x9	0x9	0x9	0x9	0x9	0x9

Table 4: Bit lengths for the 32 Huffman-encoded LengthOfMatch codes

For example, from table 4, it can be determined that the 0th Huffman-encoded LengthOfMatch code has a length of 4 bits while the 21st Huffman-encoded LengthOfMatch code has a length of 8 bits.

Using the canonical Huffman algorithm (for more information, see [CANONHUFF]), the Huffman codebook table shown in table 5 (HuffCodeL) can be obtained. The bit lengths in table 4 MUST be used to mask out the appropriate bits.

Code index	0	1	2	3	4	5	6	7
0	0x0001	0x0000	0x0002	0x0009	0x0006	0x0005	0x000d	0x000b
8	0x0003	0x001b	0x0007	0x0017	0x0037	0x000f	0x004f	0x006f
16	0x002f	0x00ef	0x001f	0x005f	0x015f	0x009f	0x00df	0x01df
24	0x003f	0x013f	0x00bf	0x01bf	0x007f	0x017f	0x00ff	0x01ff

Table 5: Huffman codebook for the 32 Huffman-encoded LengthOfMatch codes

For example: From table 5, it can be determined that the 0th Huffman-encoded LengthOfMatch code has a value of 0x0001. Applying the bit length information from table 4 (4 bits), it can be determined that the Huffman code in binary MUST be 0001.

Consider another example: From table 5, it can be determined that the 21st Huffman-encoded LengthOfMatch code has a value of 0x009f. Applying the bit length information from table 4 (8 bits), it can be determined that the Huffman code in binary MUST be 10011111.

The lookup tables in table 6 are used during encoding and decoding of the length-of-match for a given copy-tuple to determine the extra bits that will follow the LengthOfMatch code.

Index	LoMBitsLUT	LoMBaseLUT
0	0	2
1	0	3
2	0	4
3	0	5
4	0	6
5	0	7
6	0	8
7	0	9
8	1	10
9	1	12
10	1	14
11	1	16
12	2	18
13	2	22
14	2	26
15	2	30
16	3	34
17	3	42
18	3	50
19	3	58
20	4	66
21	4	82
22	4	98
23	4	114

Index	LoMBitsLUT	LoMBaseLUT
24	6	130
25	6	194
26	8	258
27	8	514
28	14	2
29	14	2

Table 6: Bit count and base value lookup tables to encode and decode length-of-match values

3.1.8.4.3 Encoding the Logically Compressed Stream

The flow chart below describes how literals and copy-tuples in a logically compressed stream are encoded.

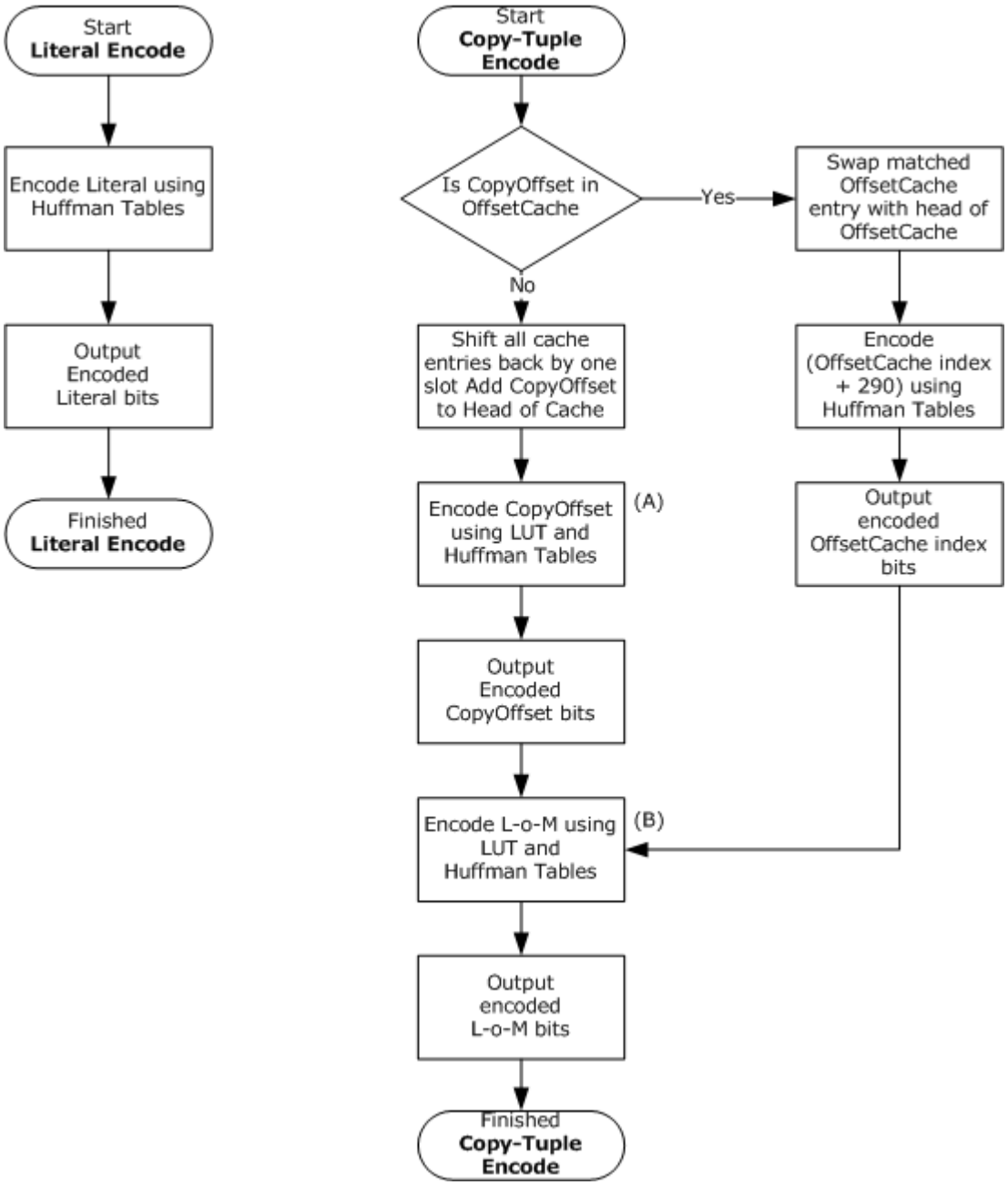


Figure 9: Encoding a logically compressed stream

Literals are merely encoded using the Huffman Tables HuffCodeLEC (table 2) and HuffLengthL (table 1). Copy-tuples are encoded using the four lookup tables in table 3 and table 6 (CopyOffsetBaseLUT, CopyOffsetBitsLUT, LoMBaseLUT, and LoMBitsLUT) and the Huffman Tables (for the algorithmic details, see sections 3.1.8.4.3.1 and 3.1.8.4.3.2).

3.1.8.4.3.1 Encoding the Copy-Offset

Encoding of the copy-offset is shown in figure 9 by the Action A item. The algorithm for encoding the copy-offset is described below:

```
LUTIndex = IndexOfEqualOrSmallerEntry(CopyOffset + 1, CopyOffsetBaseLUT)
HuffmanIndex = LUTIndex + 257
OutputBits(HuffCodeLEC[HuffmanIndex], HuffLengthLEC[HuffmanIndex])

ExtraBitsLength = CopyOffsetBitsLUT[LUTIndex]
ExtraBits = CopyOffset & ((2 ^ ExtraBitsLen) - 1)
OutputBits(ExtraBits, ExtraBitsLength)
```

The IndexOfEqualOrSmallerEntry function searches through the specified LUT table and returns the index of the entry that contains a value of equal or lesser value than the first parameter. The OutputBits function outputs the bits specified by the first parameter in the appropriate order (the number of bits to output is given by the second parameter). "^" is the exponentiation operator while "&" is the bitwise AND operator.

3.1.8.4.3.1.1 Examples of Copy-Offset Encoding

1. Encoding a copy-offset of 8:

```
LUTIndex = 6
HuffmanIndex = 6 + 257 = 263
OutputBits(0x0034, 6)

ExtraBitsLength = 2
ExtraBits = 8 & ((2 ^ 2) - 1) = 8 & 3 = bin:1000 & bin:0011 = 0
OutputBits(0x00, 2)
```

2. Encoding a copy-offset of 11:

```
LUTIndex = 6
HuffmanIndex = 6 + 257 = 263
OutputBits(0x0034, 6)

ExtraBitsLength = 2
ExtraBits = 11 & ((2 ^ 2) - 1) = 11 & 3 = bin:1011 & bin:0011 = 3
OutputBits(0x03, 2)
```

3.1.8.4.3.2 Encoding the Length-of-Match

Encoding of the length-of-match is shown in figure 9 by the Action B item. The algorithm for encoding the length-of-match is described below:

```
LUTIndex = IndexOfEqualOrSmallerEntry(LoM, LoMBaseLUT)
OutputBits(HuffCodeL[LUTIndex], HuffLengthL[LUTIndex])
ExtraBitsLength = LoMBitsLUT[LUTIndex]
ExtraBits = (LoM - 2) & ((2 ^ ExtraBitsLength) - 1)
OutputBits (ExtraBits, ExtraBitsLength)
```

The definitions of the functions used in this pseudocode are the same as those described in section 3.1.8.4.3.1.

3.1.8.4.3.2.1 Examples of Length-of-Match Encoding

1. Encoding a length-of-match of 2 bytes:

```
LUTIndex = 0
HuffmanCode = 0x0001
OutputBits(0x0001, 4)

ExtraBitsLength = 0
No extra bits to output
```

2. Encoding a length-of-match of 117 bytes:

```
LUTIndex = 23
HuffmanCode = 0x01df
OutputBits(0x01df, 9)

ExtraBitsLength = 4
ExtraBits = (117 - 2) & ((2 ^ 4) - 1) = 115 & 15 = bin:1110011 & bin:1111 = 3
OutputBits(0x03, 4)
```

3.1.8.4.4 Decoding a Compressed Stream

The flow chart below describes how the data in a compressed stream is decoded.

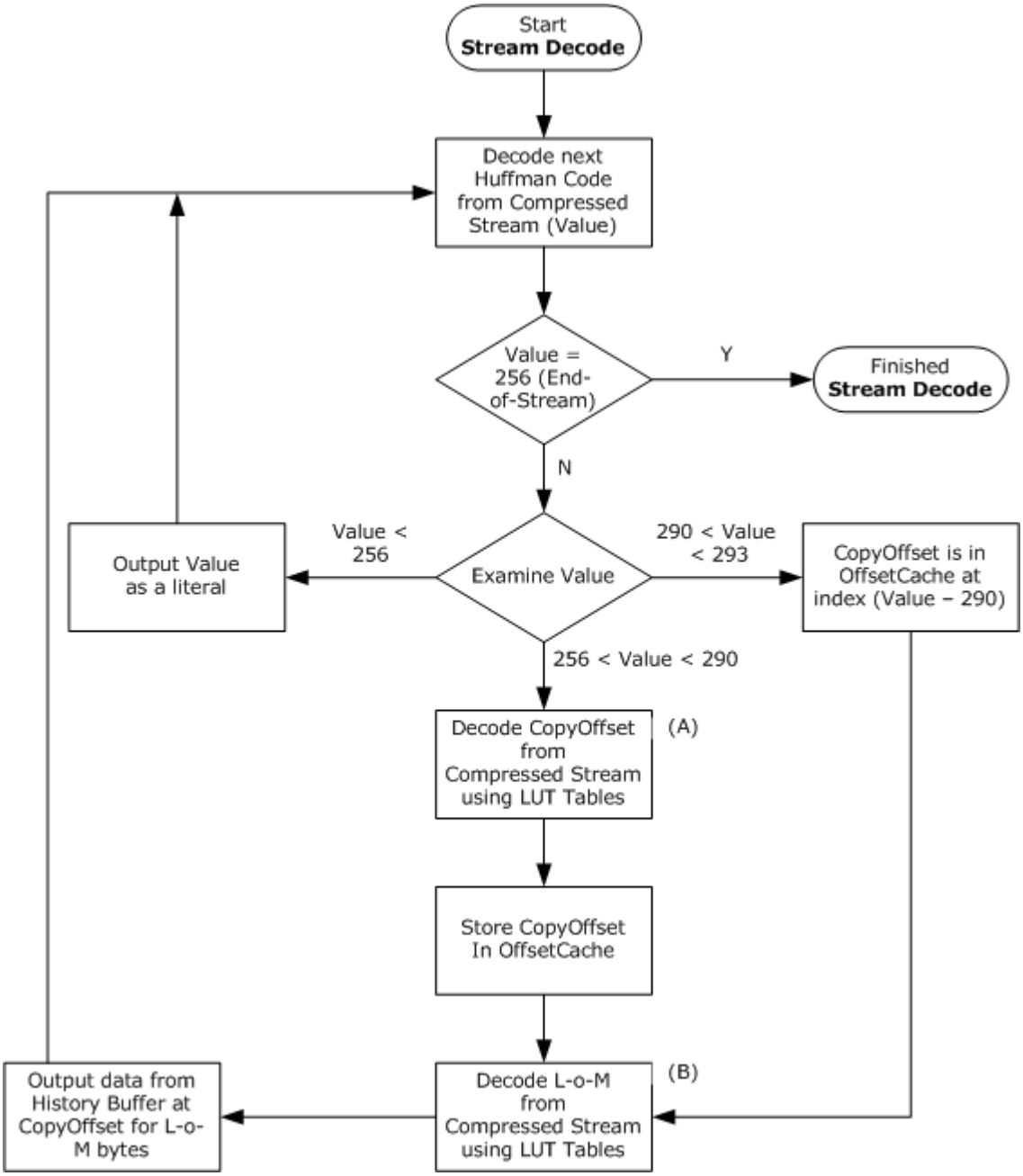


Figure 10: Decoding a compressed stream

Decoded literals are merely placed on to the output stream. However, decoded values representing copy-offset and length-of-match items require further processing using the four lookup tables in table 3 and table 6 (CopyOffsetBaseLUT, CopyOffsetBitsLUT, LoMBaseLUT, and LoMBitsLUT) (for the algorithmic details, see sections 3.1.8.4.4.1 and 3.1.8.4.4.2).

3.1.8.4.4.1 Decoding the Copy-Offset

Decoding of the copy-offset is shown in figure 10 by the Action A item. The algorithm for decoding a copy-offset is described below:

```
LUTIndex = DecodedHuffmanCode - 257
BaseLUT = CopyOffsetBaseLUT[LUTIndex]
BitsLUT = CopyOffsetBitsLUT[LUTIndex]
StreamBits = ReadBitsFromCompressedStream(BitsLUT)
CopyOffset = BaseLUT + StreamBits - 1
```

The ReadBitsFromCompressedStream function reads the number of bits specified by the parameter from the compressed stream.

3.1.8.4.4.1.1 Examples of Copy-Offset Decoding

1. Decoding a Huffman code of 0x34 followed by 2 bits of value 0:

```
DecodedHuffmanCode = 263
LUTIndex = 263 - 257 = 6
BaseLUT = 9
BitsLUT = 2

StreamBits = ReadBitsFromCompressedStream(2) = 0x00
CopyOffset = 9 + 0 - 1 = 8
```

2. Decoding a Huffman code of 0x34 followed by 2 bits of value 3:

```
DecodedHuffmanCode = 263
LUTIndex = 263 - 257 = 6
BaseLUT = 9
BitsLUT = 2

StreamBits = ReadBitsFromCompressedStream(2) = 0x03
CopyOffset = 9 + 3 - 1 = 11
```

3.1.8.4.4.2 Decoding the Length-of-Match

Decoding of the length-of-match is shown in figure 10 by the Action B item. The algorithm for decoding a length-of-match is described below:

```
HuffmanCode = ReadNextHuffmanCodeFromCompressedStream()
LUTIndex = DecodeHuffmanCode(HuffmanCode, HuffmanCodeTable)
BaseLUT = LoMBaseLUT[LUTIndex]
BitsLUT = LoMBitsLUT[LUTIndex]
StreamBits = ReadBitsFromCompressedStream(BitsLUT)
LoM = BaseLUT + StreamBits
```

The ReadNextHuffmanCodeFromCompressedStream function reads the next Huffman code from the compressed stream, and the DecodeHuffmanCode function decodes the Huffman code given by the first parameter using the Huffman codebook table specified by the second parameter. The definitions of any remaining functions used in this pseudocode are the same as those described in section 3.1.8.4.4.1.

3.1.8.4.4.2.1 Examples of Length-of-Match Decoding

1. Decoding a Huffman code of 0x0001 followed by 0 bits:

```
HuffmanCode = 0x0001
LUTIndex = 0
BaseLUT = 2
BitsLUT = 0

StreamBits = ReadBitsFromCompressedStream(0) = 0x00
LoM = 2 + 0 = 2
```

2. Decoding a Huffman code of 0x01df followed by 4 bits of value 3:

```
HuffmanCode = 0x01df
LUTIndex = 23
BaseLUT = 114
BitsLUT = 4

StreamBits = ReadBitsFromCompressedStream(4) = 0x03
LoM = 114 + 3 = 117
```

3.1.9 RDP 6.0 Bitmap Compression

RDP 6.0 Bitmap Compression is used when the RDP session color depth is 32 bits-per-pixel (bpp) and the bitmap of interest is either 24 bpp (RGB with no alpha channel) or 32 bpp (RGB with an alpha channel). The capability of a server to encode and a client to decode with RDP 6.0 Bitmap Compression is advertised in the Bitmap Capability Set (see [MS-RDPBCGR] section 2.2.7.1.2).

The main focus of RDP 6.0 Bitmap Compression is not to compress the bitmap as a single opaque block of binary data, but instead to manipulate each of the color layers as separate planes and attempt to compress them individually using a collection of compression techniques. These techniques attempt to leverage any redundancy in the color representation of the bitmaps and can additionally be followed by a simple scan-line run-length compressor if it can further compress the planes. The methods used and how they are configured is server implementation dependent.

3.1.9.1 Bitmap Compression Techniques

3.1.9.1.1 Splitting and Combining Color Planes

Planar separation is a bitmap representation in which each color plane in a bitmap is sent as separate 8 bpp bitmaps. The two color schemes used in RDP 6.0 Bitmap Compression are ARGB and AYCoCg, and these schemes have the following color planes:

Color scheme	Color planes
ARGB	Alpha, Red, Green, Blue
AYCoCg	Alpha, Luma, Orange Chroma, Green Chroma

Splitting a bitmap into constituent color planes is a straightforward operation that simply involves the transfer of data into the correct color plane. For example, the three-pixel ARGB bitmap (A1R1G1B1; A2R2G2B2; A3R3G3B3) can be split into the four color planes (A1A2A3; R1R2R3; G1G2G3; B1B2B3). In the case of 24 bpp bitmaps (or fully opaque 32 bpp bitmaps), the alpha mask for each pixel is assumed to be 0xFF, and, hence, the alpha plane may be excluded as it is trivial to reconstitute (see the DRAW_ALLOW_SKIP_ALPHA flag in [MS-RDPBCGR] section 2.2.7.1.3).

The combination of separate color planes into a single 24 bpp or 32 bpp bitmap is also a trivial operation and merely involves reconstructing the individual ARGB or AYCoCg pixels from the constituent color planes so that the component bytes are ordered correctly. For example, the four AYCoCg planes (A1A2A3; Y1Y2Y3; Co1Co2Co3; Cg1Cg2Cg3) can be reconstituted to produce the three-pixel (A1Y1Co1Cg1; A2Y2Co2Cg2; A3Y3Co3Cg3) bitmap.

3.1.9.1.2 Color Space Conversion

Color channel-based compression does not operate on the RGB color space but rather a color space based on Luminosity (Y) and differentials of two color components, namely Orange (Co) and Green (Cg). Conversion between the ARGB and AYCoCg color spaces can be performed by using forward and inverse transformations (represented using matrix multiplication). The Alpha channel (A) is never converted in either direction.

The forward transformation to convert from ARGB to AYCoCg:

```
A = A

|Y|     | 1/4   1/2    1/4|    |R|
|Co| = |  1     0     -1 | * |G|
|Cg|    |-1/2    1    -1/2|    |B|
```

The inverse transformation to convert from AYCoCg to ARGB:

```
A = A

|R|    |1   1/2   -1/2|    |Y |
|G| = |1    0     1/2| * |Co|
|B|    |1  -1/2   -1/2|    |Cg|
```

The allowed ranges of the individual color planes and their lengths in bits are as follows:

Color plane	Range	Type
Alpha (A)	0 (transparent) – 255 (opaque)	8-bits
Red (R)	0 – 255	8-bits

Color plane	Range	Type
Green (G)	0 – 255	8-bits
Blue (B)	0 – 255	8-bits
Luma (Y)	0 – 255	8-bits
Orange Chroma (Co)	-255 – 255	9-bits (two's-complement)
Green Chroma (Cg)	-255 – 255	9-bits (two's-complement)

3.1.9.1.3 Chroma Subsampling and Super-Sampling

Chroma subsampling is a compression technique that employs a lower resolution for representing the chroma planes (Co and Cg) while keeping the luminosity plane (Y) at full resolution. This method relies on the human eye being less sensitive to color than differences in luminosity. Support for chroma subsampling is advertised in the Bitmap Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

The subsampling algorithm used employs a simple averaging of four neighboring pixels in a 2x2 grid. This reduces the information needed to be sent by half in each dimension, resulting in a four-fold decrease for a two-dimensional bitmap per chroma channel. If an odd number of rows or columns are present, the subsampling is performed on a 2x1 or 1x2 grid, or not at all for a corner when the bitmap has both an odd number of rows and columns.

Super-sampling of a compressed chroma plane simply results in the expansion of a subsampled pixel into a 2x2 grid of pixels. Because this expansion results in an area with the same dimensions as the original bitmap, care must be taken to handle odd row and column counts correctly.

The figure below shows averaging (first stage) and final subsampling (second stage) being applied to a chroma plane of a 3x3 bitmap as well as the super-sampling required to reconstruct the chroma plane.

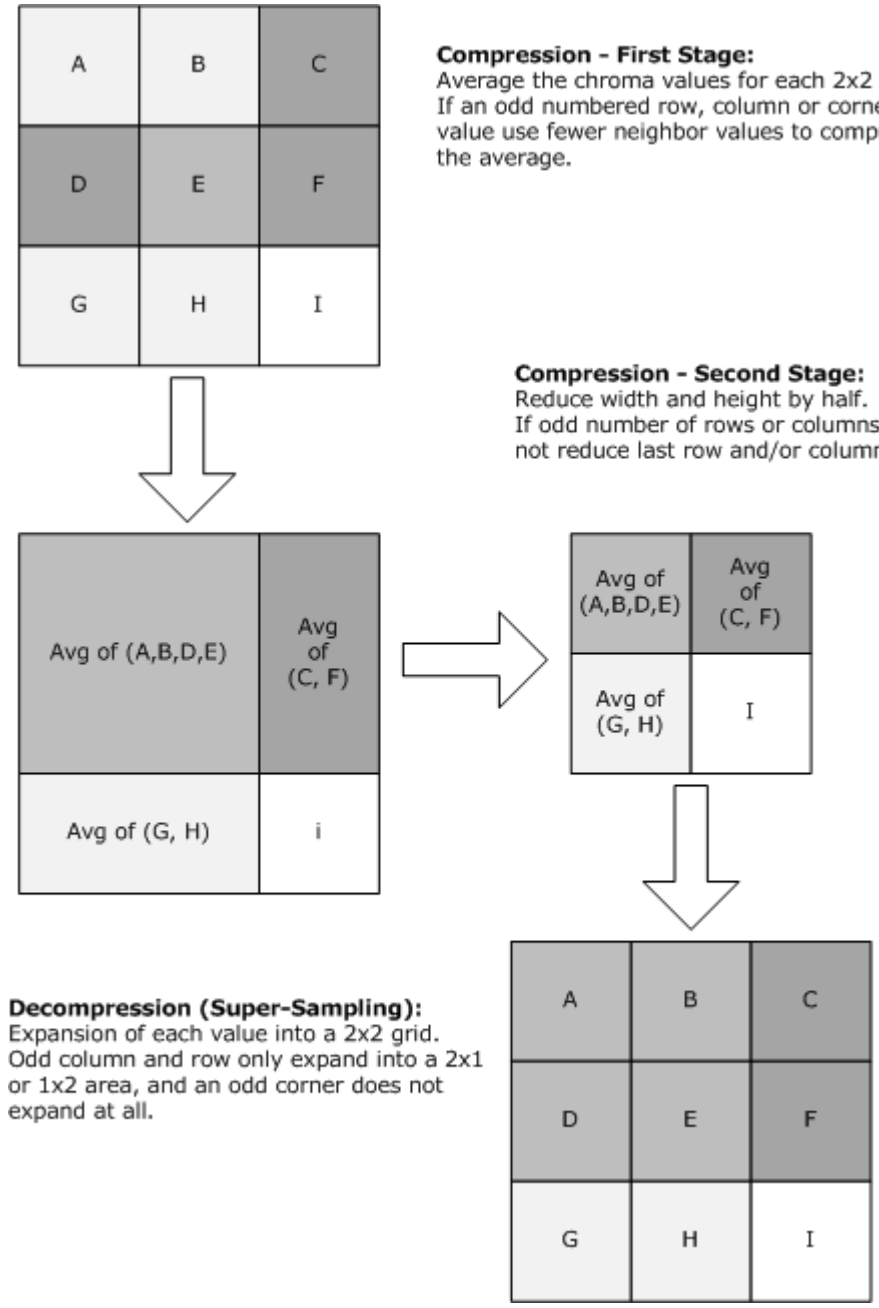


Figure 11: Chroma subsampling and super-sampling

3.1.9.1.4 Color Loss Reduction

Color Loss Reduction is a form of compression that reduces the fidelity of chroma values while maintaining the overall relative magnitude of possible chroma values and reducing the number of bits needed to represent each value. The dynamic range of the chroma representation is not reduced. This compression technique has the side-effect of reducing many similar chroma values to the same reduced value, which may improve subsequent run-length compression.

The operation to reduce chroma is simply to bit shift the chroma values toward zero while adding zeros at the high order bits. The number of bits shifted is implementation dependent and known as the Color Loss Level (CLL). The server can choose a value between 1 and 7 for the CLL. Usage of Color Loss Reduction is negotiated in the Bitmap Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

The reverse operation to recover chroma can simply be performed by shifting the reduced values back toward the high order bit and inserting zeros at the low order bit location. The client may choose to perform the reverse operation using other schemes such as a linear gradient curve as long as the final chroma values are within the ranges specified in section 3.1.9.1.2.

3.1.9.2 Run-Length Encoding

Each color plane can be individually run-length compressed. The RDP 6.0 Bitmap Compressor uses a simple scan-line compressor that breaks each scan-line into segments consisting of RAW and RUN components.

A RAW component is a non-repeating sequence in the segment while a RUN is a sequence of the last RAW value repeated run-length times. A RUN in a segment must be at least three values long (shorter sequences are encoded as RAW values). A segment will usually have both a RAW and a RUN component, but may have only one or the other, depending on the scan-line to be compressed. In the absence of a RAW component in a segment, the RAW value used to decode the RUN component is assumed to be the value zero.

For example, an initial scan-line containing the following 12 characters:

```
AAAABBCCCCCD
```

Would become:

```
RAW [A];   RUN [3] - Segment 1
RAW [BBC]; RUN [4] - Segment 2 (BB is too short to be a RUN.)
RAW [D];   RUN [0] - Segment 3 (The scan line is completed.)
```

The encoded RAW and RUN values are either absolute values (this is the case for the first scan-line) or delta values (this is the case for all other scan-lines). Delta values are relative to the previous scan-line values. The delta value calculation is performed by subtracting the previous scan-line value at the current column location from the current scan-line value being encoded.

An example of delta value usage follows (decimal values used for clarity):

```
10, 20, 30, 40, 50, 60 - First scan-line
 5, 15, 25, 35, 45, 55 - Second scan-line
 5, 15, 25, 35, 45, 55 - Third scan-line
```

Converted to delta values:

```
10, 20, 30, 40, 50, 60 - First scan-line (absolute values)
-5, -5, -5, -5, -5, -5 - Second scan-line (delta values)
```

```
 0,  0,  0,  0,  0,  0 - Third scan-line (delta values)
```

Which, converted to segments, becomes:

```
RAW [10,20,30,40,50,60]; RUN [0]
RAW [-5];                RUN [5]
RAW [<none>];            RUN [6] (Previous base value assumed to be 0.)
```

Due to the fact that the lengths of RAW and RUN components are limited to 4-bit values (see section 2.2.2.1.2), individual segments may be broken up further into subsegments during encoding to produce RUN sequences consisting of more than 16 values.

3.1.9.2.1 Encoding Run-Length Sequences

Once a run-length encoder has broken up a scan-line into segments, it MUST encode each segment as one or more subsegments. The structure of segments and subsegments is the same, and is defined in section 2.2.2.1.2.

The process of encoding scan-lines using RDP 6.0 RLE is best illustrated with a practical example. Assume the following three scan-lines are present in a bitmap:

```
255, 255, 255, 255, 254, 253 - First scan-line
254, 192, 132,  96,  75,  25 - Second scan-line
253, 140,  62,  14, 135, 193 - Third scan-line
```

Converted to delta values:

```
255, 255,  255,  255,  254,  253 - First scan-line (absolute values)
 -1, -63, -123, -159, -179, -228 - Second scan-line (delta values)
 -1, -52,  -70,  -82,   60,  168 - Third scan-line (delta values)
```

Each value MUST be interpreted as a 1-byte two's complement signed value:

```
-1,  -1,   -1,  -1, -2,  -3 - First scan-line (absolute values)
-1, -63, -123,  97, 77,  28 - Second scan-line (delta values)
-1, -52,  -70, -82, 60, -88 - Third scan-line (delta values)
```

The next step is to encode the delta values. If the delta value does not have the most significant bit set (0 – 127), shift it one bit toward the highest bit. If the delta value has the most significant bit set (-1 to -128), take the 8-bit two's complement of the value (invert the bits and add one or subtract from 256), bit shift it toward the high bit value (multiply by 2), and decrement by 1. This transformation results in:

```
-1,  -1,   -1,  -1,   -2,  -3 - First scan-line (absolute values)
```

```
 1, 125,   11, -62, -102,  56 - Second scan-line (delta values)
 1, 103, -117, -93,  120, -81 - Third scan-line (delta values)
```

Representing these signed two's complement values as 8-bit hexadecimal values:

```
0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xFD - First scan-line
0x01, 0x7D, 0xF5, 0xC2, 0x9A, 0x38 - Second scan-line
0x01, 0x67, 0x8B, 0xA3, 0x78, 0xAF - Third scan-line
```

Which, converted to RLE segments, become:

```
RAW [0xFF];                          RUN [3] - Segment 1
RAW [0xFE,0xFD];                     RUN [0] - Segment 2
RAW [0x01,0x7D,0xF5,0xC2,0x9A,0x38]; RUN [0] - Segment 3
RAW [0x01,0x67,0x8B,0xA3,0x78,0xAF]; RUN [0] - Segment 4
```

Using the structure defined in section 2.2.2.1.2, the first segment, which uses absolute values (see section 3.1.9.2), is encoded as follows:

```
controlByte: (cRawBytes = 1, nRunLength = 3); rawValues: 0xFF
```

Which yields the following byte stream for the first segment:

```
0x13, 0xFF - Segment 1
```

Encoding the remaining segments yields:

```
0x20, 0xFE, 0xFD - Segment 2
0x60, 0x01, 0x7D, 0xF5, 0xC2, 0x9A, 0x38 - Segment 3
0x60, 0x01, 0x67, 0x8B, 0xA3, 0x78, 0xAF - Segment 4
```

The flow chart below illustrates how a given set of RAW values and a RUN (which can be zero) is encoded using RDP 6.0 RLE.

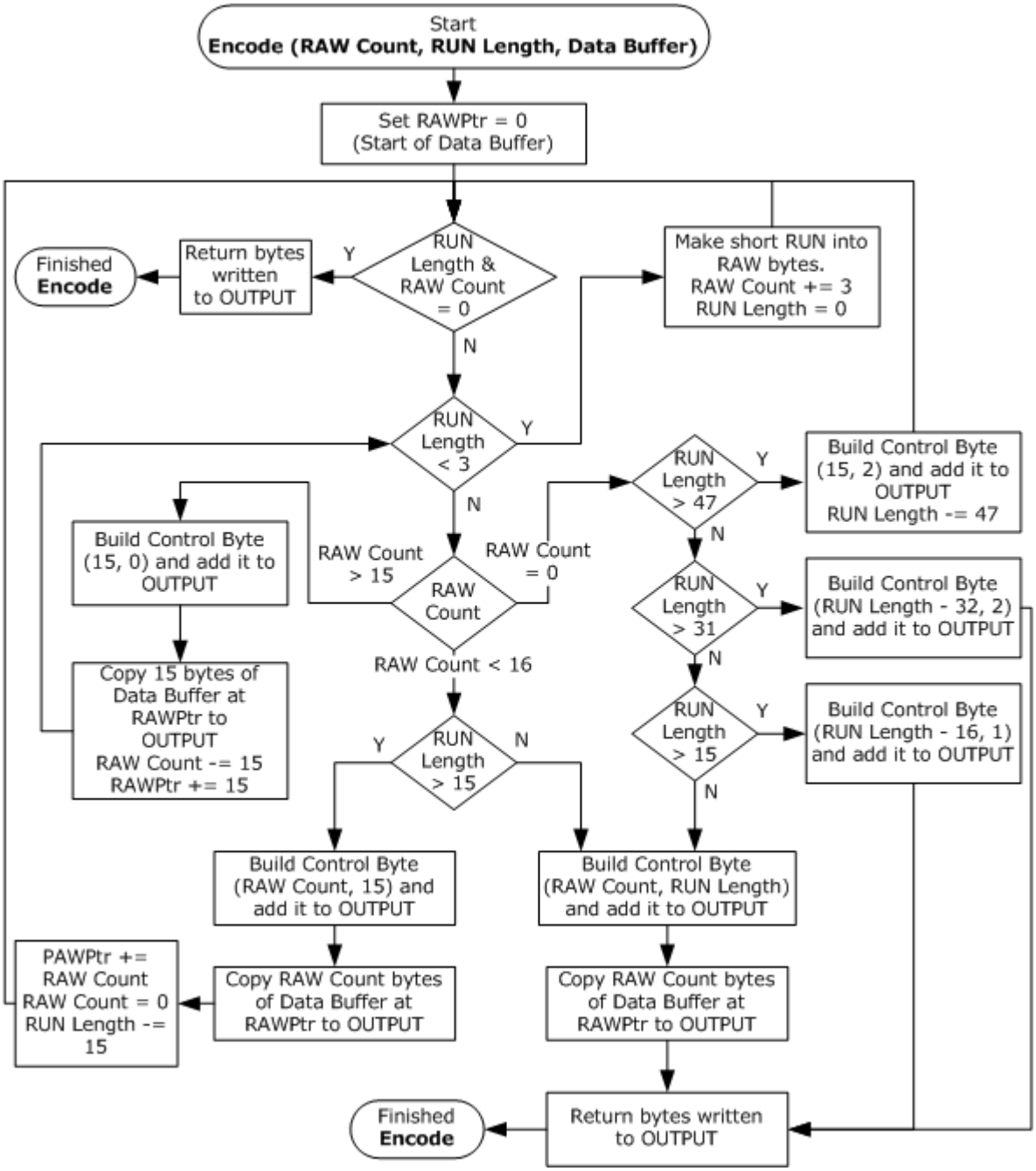


Figure 12: Encoding data using RDP 6.0 Run-Length Encoding (RLE)

3.1.9.2.2 Extra Long RUN Sequences

The lengths of RAW and RUN components are limited to 4-bit values and, as a result, encoding of run-lengths greater than 15 values requires special casing, as described in section 2.2.2.1.2.

Encoding of an extra long RUN sequence is best described with an example. Assume there is a pattern consisting of the letter A repeated 100 times. The resulting RLE segments are:

```
RAW [A];      RUN [15] - Segment 1
RAW [<none>]; RUN [47] - Segment 2
RAW [<none>]; RUN [37] - Segment 3
```

Using the structure and rules in section 2.2.2.1.2, these segments are encoded as:

```
controlByte: (1, 15); rawValues: A
controlByte: (15, 2); rawValues: <none>
controlByte: (5, 2); rawValues: <none>
```

In hexadecimal, this becomes:

```
0x1F, 0x41, 0xF2, 0x52
```

3.1.9.2.3 Decoding Run-Length Sequences

Encoding of run-length sequences ensures that there is at least one subsegment per scan-line. The control byte described in section 2.2.2.1.2 contains all of the information necessary to decode the sequence bytes.

The process of decoding an encoded sequence of bytes for a color plane using RDP 6.0 RLE is best illustrated with a practical example. Assume the following bytes are from a color plane in a bitmap that is six pixels wide by three pixels high:

```
0x13, 0xFF, 0x20, 0xFE, 0xFD, 0x60, 0x01, 0x7D,
0xF5, 0xC2, 0x9A, 0x38, 0x60, 0x01, 0x67, 0x8B,
0xA3, 0x78, 0xAF
```

The first byte is the control byte of the first segment from the first scan-line:

```
0x13 -> Control byte = 1 RAW value, 3-length RUN of the last raw value
0xFF -> Raw value
```

Hence, decoding the first segment according to the information in the control byte yields:

```
0xFF, 0xFF, 0xFF, 0xFF
```

Because these values are from the first scan-line, the values are absolute. Shown in unsigned decimal, the values are:

```
255, 255, 255, 255
```

The second segment control byte is:

```
0x20
```

This implies two raw values and a zero-length run, resulting in the following two unsigned decimal absolute values:

```
0xFE, 0xFD
```

This completes the first scan-line, and all values decoded from this point are deltas and MUST be specially decoded. Before decoding, the deltas MUST be interpreted as unsigned 1-byte values. If the encoded delta value is odd, decrement it by 1, shift it one bit toward the lowest bit, and subtract it from 255. This yields the original absolute value. If the encoded delta value is even, shift it one bit toward the lowest bit. This yields the original absolute value.

Examining the third segment yields six raw values and a zero-length run:

```
0x01, 0x7D, 0xF5, 0xC2, 0x9A, 0x38
```

The deltas MUST be decoded. The unsigned decimal representation of the deltas is as follows:

```
1, 125, 245, 194, 154, 56
```

Applying the delta transformation to the deltas in the second scan-line yields the following 1-byte unsigned values:

```
255, 193, 133, 97, 77, 28
```

To compute the final absolute values for the second row, the unsigned 8-bit delta values are added to the absolute values of the first scan-line using 1-byte arithmetic:

```
Column 1: 255 + 255 = 254
Column 2: 255 + 193 = 192
Column 3: 255 + 133 = 132
Column 4: 255 +  97 =  96
Column 5: 254 +  77 =  75
Column 6: 253 +  28 =  25
```

This completes the second scan-line, and the final absolute values are:

```
254, 192, 132, 96, 75, 25
```

The third segment has six raw values and a zero-length run:

```
0x01, 0x67, 0x8B, 0xA3, 0x78, 0xAF
```

The decoded unsigned 8-bit unsigned values representing the final deltas are as follows:

```
255, 204, 186, 174, 60, 168
```

To compute the final absolute values for the third row, the unsigned 8-bit delta values are added to the absolute values of the second scan-line using 1-byte arithmetic:

```
Column 1: 254 + 255 = 253
Column 2: 192 + 204 = 140
Column 3: 132 + 186 =  62
Column 4:  96 + 174 =  14
Column 5:  75 +  60 = 135
Column 6:  25 + 168 = 193
```

This completes the second scan-line, and the final absolute values are:

```
253, 140, 62, 14, 135, 193
```

The fully decoded plane is:

```
255, 255, 255, 255, 254, 253
254, 192, 132,  96,  75,  25
253, 140,  62,  14, 135, 193
```

3.1.9.3 Compressing a Bitmap

The overall scheme used to compress a bitmap with RDP 6.0 Bitmap Compression is described in the following figure. This figure effectively shows how the techniques described in section 3.1.9.1 are employed. The usage of color reduction and chroma subsampling is negotiated in the Bitmap Capability Set (see [MS-RDPBCGR] section 2.2.7.1.2). The color loss level to use (see section 3.1.9.1.4) is not negotiated and is server-implemention dependent.

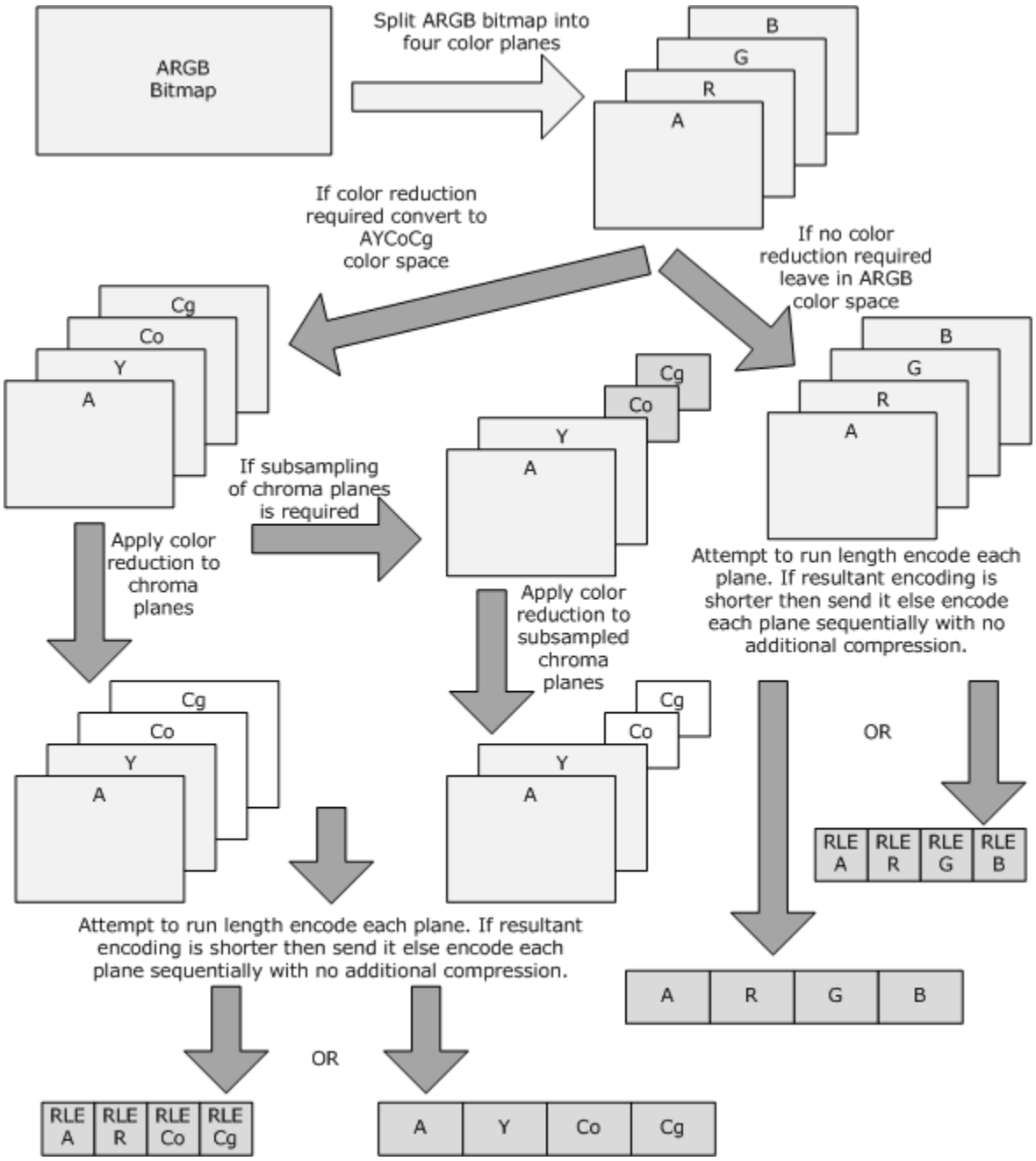


Figure 13: Compressing a bitmap using RDP 6.0 Bitmap Compression

3.1.9.4 Decompressing a Bitmap

The decision-flow used to decompress a bitmap that is compressed with RDP 6.0 Bitmap Compression is illustrated in the following figure. The flags describing the bitmap data (present in the compressed data format header) are defined in section 2.2.2.1.

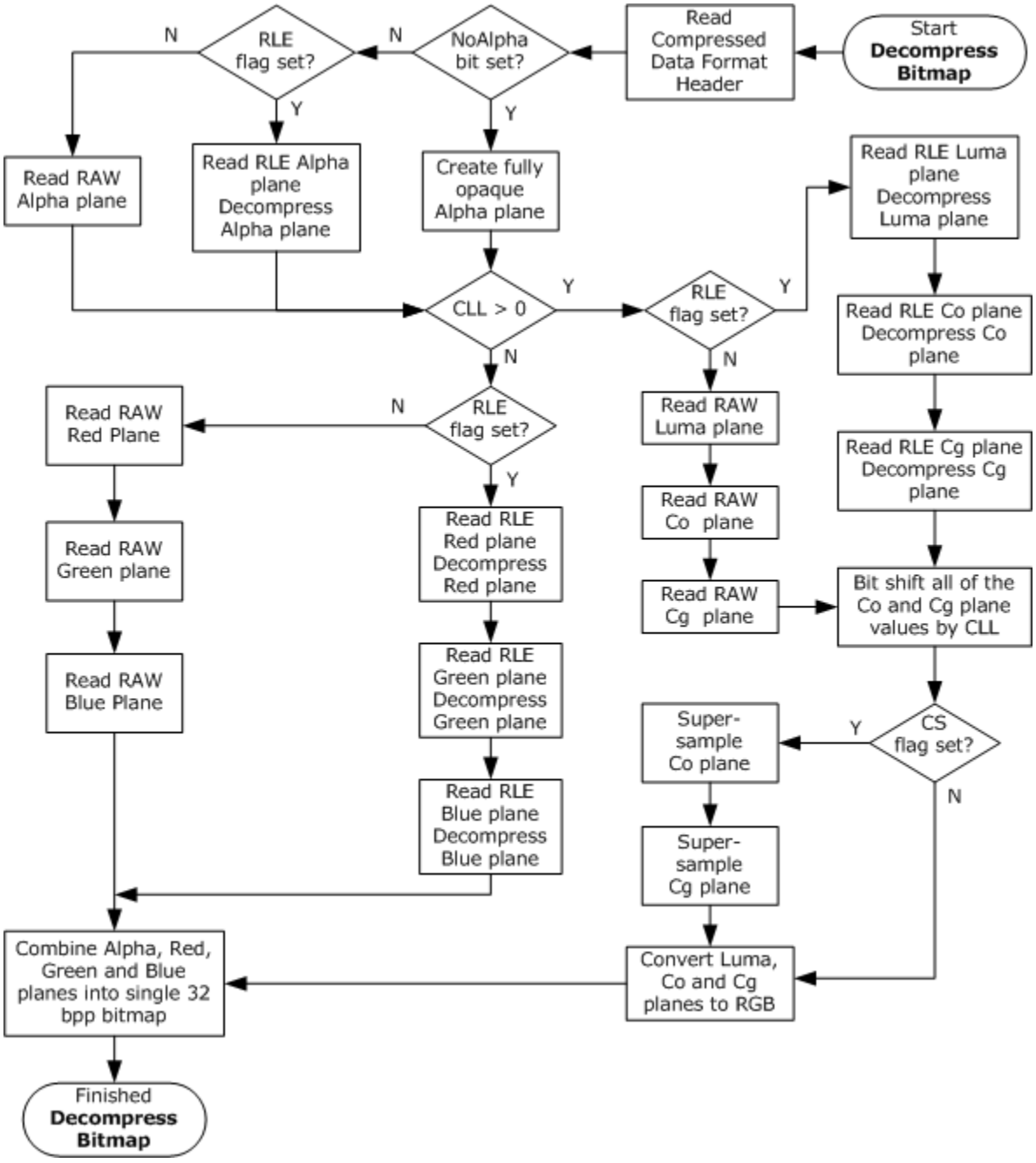


Figure 14: Decompression of a bitmap compressed with RDP 6.0 Bitmap Compression

3.2 Client Details

3.2.1 Abstract Data Model

No client abstract data model is specified. For the common conceptual data model, see section 3.1.1.

3.2.1.1 Primary Drawing Order History

The Primary Drawing Order History store holds information on the fields that have been received in primary drawing orders. There are three pieces of information that MUST be recorded:

- Last primary order type processed.
- Current bounding rectangle.
- Per-order record of the last value received in each field.

These records are updated as each primary drawing order is processed, and are used to efficiently decode and process primary drawing orders received from the server (see section 3.2.5.1.1.1).

3.2.1.2 Save Bitmap

The Save Bitmap is a 480 pixel by 480 pixel bitmap that is used to temporarily store desktop bitmap images received in the SaveBitmap Primary Drawing Order (see section 3.2.5.1.1.1.12). Bitmaps stored in the Save Bitmap are tiled to maximize the number of bitmaps that can be stored (a graphical representation of how bitmaps are stored is shown in figure 6). If support for the SaveBitmap Primary Drawing Order is not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3), the Save Bitmap is not required.

3.2.2 Timers

No timers are used.

3.2.3 Initialization

No initialization steps are specified.

3.2.4 Higher-Layer Triggered Events

No client higher-layer triggered events are used.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Drawing Orders

All drawing orders are encapsulated in an Orders Update (section 2.2.2.2), which is received as part of the Graphics Update PDU (see [MS-RDPBCGR] section 2.2.9.1.1.3); or they are encapsulated in a Fast-Path Orders Update (section 2.2.2.3), which is received as part of the Fast-Path Update PDU (see [MS-RDPBCGR] section 2.2.9.1.2.1).

There are three classes of drawing orders:

- Primary Drawing Orders (section 2.2.2.3.1.1)
- Secondary Drawing Orders (section 2.2.2.3.1.2)
- Alternate Secondary Drawing Orders (section 2.2.2.3.1.3)

Orders belonging to each of these classes are packed together into an Orders Update structure or a Fast-Path Orders Update structure, each order being aligned on a byte boundary.

3.2.5.1.1 Primary Drawing Orders

3.2.5.1.1.1 Processing Primary Drawing Orders

All primary drawing orders MUST conform to the structure and rules defined in section 2.2.2.3.1.1.2.

To efficiently decode and process a primary drawing order, the client MUST use a Primary Drawing Order History (section 3.2.1.1) store. This store holds three pieces of information:

- Last primary order type processed.
- Current bounding rectangle.
- Per-order record of the last value received for each field.

These stored records allow the client to correctly decode a received primary drawing order, as only fields that have new values are required to be included in orders sent by the server; if a field is unchanged from the value that it had when the order type was last sent, it SHOULD NOT be included in the order. The fields that are present in the order MUST be indicated by the fieldFlags field.

If all of the Coord-type fields (see section 2.2.2.3.1.1.1.1) in an order can be represented as a signed delta in the range -127 to 128 from the previous field value, all of these fields SHOULD contain delta-coordinates (see sections 2.2.2.3.1.1.1.1 and 2.2.2.3.1.1.1.2). The presence of delta-coordinates MUST be indicated by the TS_DELTA_COORDINATES (0x10) flag in the primary drawing header.

The basic process to decode a primary drawing order begins with reading the controlFlags header byte (this is common to all drawing classes) to determine the order class. If order is a primary drawing order, the following sequence of steps SHOULD be used to decode the order:

1. The control flags extract the following information:
   - Whether or not the order type is specified.
   - The fields that are present in the primary order field data.
   - Whether or not any components of the bounding rectangle are present.
   - Whether or not delta-coordinates are used in the primary order Coord-type fields.
2. If support for the primary drawing order was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3), the client SHOULD ignore the order, and processing SHOULD cease.
3. Determine whether or not all of the data required to decode and process the order has been received in the Orders Update (section 2.2.2.2) structure or the Fast-Path Orders Update (section 2.2.2.3) structure. If the packet does not contain enough data, processing SHOULD cease, and the order update SHOULD be ignored.
4. Read the order data, and validate the fields to make sure all the field data is consistent with the order specification (for example, the maximum number of fields and maximum order size MUST conform to the order specification). If any of the field data for a given order is inconsistent or refers to non-existent or invalid items (such as a non-existent cache entry or invalid brush format), processing of the order SHOULD terminate, and it SHOULD also be ignored.

Once the order has been decoded, and all of the information necessary to process it has been collected, the data MUST be handed off to a graphics rendering module so that the images from the

remote system can be displayed locally on the client system. The client MUST also update the records in the Primary Drawing Order History (section 3.3.1.2) to ensure that future orders can be decoded correctly.

3.2.5.1.1.1.1 Processing of DstBlt

The structure and fields of the DstBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.1, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.2 Processing of MultiDstBlt

The structure and fields of the MultiDstBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.2, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.3 Processing of PatBlt

The structure and fields of the PatBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.3, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If a cached brush is specified in this order, that brush MUST have been received by the client in a prior Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6). If this is not the case, the client SHOULD ignore this order. Furthermore, if support for brush caching was not specified in the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8), and a cached brush is included in the PatBlt order, the client SHOULD ignore this order.

3.2.5.1.1.1.4 Processing of MultiPatBlt

The structure and fields of the MultiPatBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.4, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If a cached brush is specified in this order, that brush MUST have been received by the client in a prior Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6). If this is not the case, the client SHOULD ignore this order. Furthermore, if support for brush caching was not specified in the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8), and a cached brush is included in the MultiPatBlt order, the client SHOULD ignore this order.

3.2.5.1.1.1.5 Processing of OpaqueRect

The structure and fields of the OpaqueRect Primary Drawing Order are specified in section 2.2.2.3.1.1.2.5, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.6 Processing of MultiOpaqueRect

The structure and fields of the MultiOpaqueRect Primary Drawing Order are specified in section 2.2.2.3.1.1.2.6, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.7 Processing of ScrBlt

The structure and fields of the ScrBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.7, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.8 Processing of MultiScrBlt

The structure and fields of the MultiScrBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.8, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.9 Processing of MemBlt

The structure and fields of the MemBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.9, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

The source bitmap associated with the MemBlt order MUST reside in one of two possible locations:

1. One of the negotiated bitmap caches (see section 3.1.1.1.1).
2. Offscreen Bitmap Cache (section 3.1.1.1.5).

If the bitmap cache ID (specified as part of the **cacheId** field) refers to one of the negotiated bitmap caches, the actual bitmap data MUST have been received in a Revision 1 or 2 Cache Bitmap Secondary Drawing Order (see sections 3.2.5.1.2.1.1 and 3.2.5.1.2.1.2). If the **cacheIndex** field is set to BITMAPCACHE_WAITING_LIST_INDEX (32767), the last bitmap cache entry MUST contain the source bitmap—this implies that the bitmap data was received in a Cache Bitmap (Revision 2) order, and the CBR2_DO_NOT_CACHE (0x10) flag was set.

However, if the bitmap cache ID is set to TS_BITMAPCACHE_SCREEN_ID (0xFF), the source bitmap resides in the Offscreen Bitmap Cache at the entry location specified by the **cacheIndex** field. This entry MUST have been created and initialized in response to the processing of prior Create Offscreen Bitmap (see section 3.2.5.1.3.1.1) and Switch Surface (see section 3.2.5.1.3.1.2) Alternate Secondary Drawing Orders; the Create Offscreen Bitmap Order creates the offscreen bitmap while the Switch Surface Order is used to redirect all drawing operations to the offscreen bitmap.

3.2.5.1.1.1.10 Processing of Mem3Blt

The structure and fields of the Mem3Blt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.10, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

The decoding and processing of the Mem3Blt Order follow the same principles as those outlined for the MemBlt Order specified in section 3.2.5.1.1.1.9. However, the Mem3Blt Order includes fields to specify a brush.

If a cached brush is specified in this order, that brush MUST have been received by the client in a prior Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6). If this is not the case, the client SHOULD ignore this order. Furthermore, if support for brush caching was not specified in the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3), and a cached brush is included in the Mem3Blt Order, the client SHOULD ignore this order.

3.2.5.1.1.1.11 Processing of LineTo

The structure and fields of the LineTo Primary Drawing Order are specified in section 2.2.2.3.1.1.2.11, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.12 Processing of SaveBitmap

The structure and fields of the SaveBitmap Primary Drawing Order are specified in section 2.2.2.3.1.1.2.12, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order. Bitmap data received in this order MUST be stored in the Save Bitmap (see section 3.2.1.2).

3.2.5.1.1.1.13 Processing of GlyphIndex

The structure and fields of the GlyphIndex Primary Drawing Order are specified in section 2.2.2.3.1.2.5, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If support for glyph caching was not negotiated using the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9), the client SHOULD ignore this order as it requires the existence of the glyph caches (see section 3.1.1.1.2).

All of the glyphs associated with the glyph cache indices specified in the order MUST have been received by the client in a prior Revision 1 or 2 Cache Glyph Secondary Drawing Order (see sections 3.2.5.1.2.1.4 and 3.2.5.1.2.1.5), or a FastGlyph Primary Drawing Order (see section 3.2.5.1.1.1.15). If a fragment cache index is specified in a USE clause, that fragment MUST have been cached while processing an ADD clause in a prior GlyphIndex or FastIndex Order.

If a cached brush is specified in this order, that brush MUST have been received by the client in a prior Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6). If this is not the case, the client SHOULD ignore this order. Furthermore, if support for brush caching was not specified in the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8), and a cached brush is included in the GlyphIndex order, the client SHOULD ignore this order.

3.2.5.1.1.1.14 Processing of FastIndex

The structure and fields of the FastIndex Primary Drawing Order are specified in section 2.2.2.3.1.1.2.14, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

The decoding and processing of the FastIndex Order follow the same principles as those outlined for the GlyphIndex Order specified in section 3.2.5.1.1.1.13. However, the FastIndex Order does not use cached brushes and also utilizes a more efficient field encoding.

3.2.5.1.1.1.15 Processing of FastGlyph

The structure and fields of the FastGlyph Primary Drawing Order are specified in section 2.2.2.3.1.1.2.15, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If support for glyph caching was not negotiated using the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9), the client SHOULD ignore this order as it requires the existence of the glyph caches (see section 3.1.1.1.2).

All of the glyphs associated with the glyph cache indices specified in the order MUST have been received by the client in a prior Revision 1 or 2 Cache Glyph Secondary Drawing Order (see sections 3.2.5.1.2.1.4 and 3.2.5.1.2.1.5) or a FastGlyph Primary Drawing Order.

Once the client has completed decoding and processing the FastGlyph order, and the glyph has been rendered successfully, the glyph data MUST be stored in the specified glyph cache.

3.2.5.1.1.1.16 Processing of PolygonSC

The structure and fields of the PolygonSC Primary Drawing Order are specified in section 2.2.2.3.1.1.2.16, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.17 Processing of PolygonCB

The structure and fields of the PolygonCB Primary Drawing Order are specified in section 2.2.2.3.1.1.2.17, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If a cached brush is specified in this order, that brush MUST have been received by the client in a prior Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6). If this is not the case, the client SHOULD ignore this order. Furthermore, if support for brush caching was not specified in the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8), and a cached brush is included in the PolygonCB order, the client SHOULD ignore this order.

3.2.5.1.1.1.18 Processing of Polyline

The structure and fields of the Polyline Primary Drawing Order are specified in section 2.2.2.3.1.1.2.18, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.19 Processing of EllipseSC

The structure and fields of the EllipseSC Primary Drawing Order are specified in section 2.2.2.3.1.1.2.19, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

3.2.5.1.1.1.20 Processing of EllipseCB

The structure and fields of the EllipseCB Primary Drawing Order are specified in section 2.2.2.3.1.1.2.20, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If a cached brush is specified in this order, that brush MUST have been received by the client in a prior Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6). If this is not the case, the client SHOULD ignore this order. Furthermore, if support for brush caching was not specified in the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8), and a cached brush is included in the EllipseCB order, the client SHOULD ignore this order.

3.2.5.1.1.1.21 Processing of DrawNineGrid

The structure and fields of the DrawNineGrid Primary Drawing Order are specified in section 2.2.2.3.1.1.2.21, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

If support for NineGrid bitmap caching was not negotiated using the DrawNineGrid Cache Capability Set (see section 2.2.1.2), the client SHOULD ignore the order since the order refers to NineGrid bitmaps in the NineGrid Bitmap Cache (see section 3.1.1.1.6).

The source bitmap (which resides in the NineGrid Bitmap Cache) MUST have been created and initialized as a result of processing prior Create NineGrid Bitmap (see section 3.2.5.1.3.1.3) and Stream Bitmap (see section 3.2.5.1.3.1.4) Alternate Secondary Orders.

3.2.5.1.1.1.22 Processing of MultiDrawNineGrid

The structure and fields of the MultiDrawNineGrid Primary Drawing Order are specified in section 2.2.2.3.1.1.2.22, and the techniques described in section 3.2.5.1.1.1 demonstrate how to decode and process the order.

The decoding and processing of the MultiDrawNineGrid Order follow the same principles as those outlined for the DrawNineGrid order specified in section 3.2.5.1.1.1.21. However, the MultiDrawNineGrid includes multiple clipping rectangles, as opposed to the DrawNineGrid order, which only includes one clipping rectangle.

3.2.5.1.2 Secondary Drawing Orders

3.2.5.1.2.1 Processing Secondary Drawing Orders

All secondary drawing orders are identified by the Secondary Drawing Order Header (section 2.2.2.3.1.2.1.1) and are used to manipulate the RDP caches (see section 3.1.1.1).

- The Cache Bitmap (Revision 1) Secondary Drawing Order (see section 3.2.5.1.2.1.1) manages the Revision 1 Bitmap Cache (see section 3.1.1.1.1).
- The Cache Bitmap (Revision 2) Secondary Drawing Order (see section 3.2.5.1.2.1.2) manages the Revision 2 Bitmap Cache (see section 3.1.1.1.1).
- The Cache Color Table Secondary Drawing Order (see section 3.2.5.1.2.1.3) manages the Color Table Cache (see section 3.1.1.1.3).
- The Revision 1 and 2 Cache Glyph Secondary Drawing Orders (see sections 3.2.5.1.2.1.4 and 3.2.5.1.2.1.5) manage the Glyph Cache (see section 3.1.1.1.2).
- The Cache Brush Secondary Drawing Order (see section 3.2.5.1.2.1.6) manages the Brush Cache (see section 3.1.1.1.4).

If the client has not advertised support for a particular cache type, it SHOULD ignore any secondary and primary drawing orders associated with that cache, if they are sent by the server.

3.2.5.1.2.1.1 Processing of Cache Bitmap (Revision 1)

The structure and fields of the Cache Bitmap (Revision 1) Secondary Drawing Order are specified in section 2.2.2.3.1.2.2. The order fields MUST be processed in accordance with this description.

If the client does not support Revision 1 bitmap caching (negotiated in the Revision 1 Bitmap Cache Capability Set described in [MS-RDPBCGR] section 2.2.7.1.5.1), or the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders, this order SHOULD be ignored.

3.2.5.1.2.1.2 Processing of Cache Bitmap (Revision 2)

The structure and fields of the Cache Bitmap (Revision 2) Secondary Drawing Order are specified in section 2.2.2.3.1.2.3. The order fields MUST be processed in accordance with this description.

If a 64-bit key is included in the order, indicated by the presence of the CBR2_PERSISTENT_KEY_PRESENT (0x02) flag, the client has the option of caching the bitmap in persistent storage; the server will include a 64-bit key if the capabilities for the bitmap cache negotiated in the Bitmap Cache (Revision 2) Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.2) requested that bitmap keys be sent. The 64-bit key is a unique identifier associated with the bitmap and can be sent to the server on a reconnection as part of the Persistent Key List PDU (see [MS-RDPBCGR] section 2.2.1.17).

If the client does not support Revision 2 bitmap caching (negotiated in the Bitmap Cache (Revision 2) Capability Set described in [MS-RDPBCGR] section 2.2.7.1.5.2), or the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders, this order SHOULD be ignored.

3.2.5.1.2.1.3 Processing of Cache Color Table

The structure and fields of the Cache Color Table Secondary Drawing Order are specified in section 2.2.2.3.1.2.4. The order fields MUST be processed in accordance with this description.

If the client does not support the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders, his order SHOULD be ignored.

3.2.5.1.2.1.4 Processing of Cache Glyph (Revision 1)

The structure and fields of the Cache Glyph (Revision 1) Secondary Drawing Order are specified in section 2.2.2.3.1.2.5. The order fields MUST be processed in accordance with this description.

If the client does not support glyph caching (negotiated in the Glyph Cache Capability Set described in [MS-RDPBCGR] section 2.2.7.1.9) or the GlyphIndex (section 2.2.2.3.1.2.5), FastIndex (section 2.2.2.3.1.1.2.14), and FastGlyph (section 2.2.2.3.1.1.2.15) Primary Drawing Orders, this order SHOULD be ignored.

3.2.5.1.2.1.5 Processing of Cache Glyph (Revision 2)

The structure and fields of the Cache Glyph (Revision 2) Secondary Drawing Order are specified in section 2.2.2.3.1.2.6. The order fields MUST be processed in accordance with this description.

The decoding and processing of the Cache Glyph (Revision 2) order follows the same principles as those outlined for the Cache Glyph (Revision 1) order specified in section 3.2.5.1.2.1.4. However, the Revision 2 order utilizes a more efficient field encoding.

3.2.5.1.2.1.6 Processing of Cache Brush

The structure and fields of the Cache Brush Secondary Drawing Order are specified in section 2.2.2.3.1.2.7. The order fields MUST be processed in accordance with this description.

The primary drawing orders that use cached brushes are:

- PatBlt (see section 3.2.5.1.1.1.3)
- MultiPatBlt (see section 3.2.5.1.1.1.4)

- Mem3Blt (see section 3.2.5.1.1.1.10)
- GlyphIndex (see section 3.2.5.1.1.1.13)
- PolygonCB (see section 3.2.5.1.1.1.17)
- EllipseCB (see section 3.2.5.1.1.1.20)

If the client does not support brush caching (negotiated in the Brush Capability Set described in [MS-RDPBCGR] section 2.2.7.1.8) or any primary drawing orders that use cached brushes, this order SHOULD be ignored.

3.2.5.1.3 Alternate Secondary Drawing Orders

3.2.5.1.3.1 Processing Alternate Secondary Drawing Orders

All alternate secondary drawing orders are identified by the Alternate Secondary Drawing Order Header (section 2.2.2.3.1.3.1.1), and are used to manage offscreen and NineGrid bitmaps and to transport opaque GDI+ 1.1 records.

- The Create Offscreen Bitmap (see section 3.2.5.1.3.1.1) and Switch Surface (see section 3.2.5.1.3.1.2) Alternate Secondary Drawing Orders are used to manipulate the bitmaps in the Offscreen Bitmap Cache (see section 3.1.1.1.5).
- The Create NineGrid Bitmap (see section 3.2.5.1.3.1.3) and Stream Bitmap (see section 3.2.5.1.3.1.4) Alternate Secondary Drawing Orders are used to manipulate the bitmaps in the NineGrid Bitmap Cache (see section 3.1.1.1.6).
- The GDI+ Alternate Secondary Drawing Orders (see section 2.2.2.3.1.3.6) are used to transport opaque GDI+ 1.1 records for processing or placement in the GDI+ Caches (see section 3.1.1.1.7).

If the client has not advertised support for a particular cache type or feature, it SHOULD ignore any alternate secondary and primary drawing orders associated with that cache or feature, if they are sent by the server.

3.2.5.1.3.1.1 Processing of Create Offscreen Bitmap

The structure and fields of the Create Offscreen Bitmap Alternate Secondary Drawing Order are specified in section 2.2.2.3.1.3.2. The order fields MUST be processed in accordance with this description.

If the client does not support offscreen bitmaps (negotiated in the Offscreen Bitmap Cache Capability Set described in [MS-RDPBCGR] section 2.2.7.1.10), or the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders, this order SHOULD be ignored.

3.2.5.1.3.1.2 Processing of Switch Surface

The structure and fields of the Switch Surface Alternate Secondary Drawing Order are specified in section 2.2.2.3.1.3.3. The order fields MUST be processed in accordance with this description.

The Create Offscreen Bitmap Alternate Secondary Drawing Order (see section 3.2.5.1.3.1.1), which is used to create the offscreen bitmap referenced in this order, MUST have been received by the client before processing this order.

If the client does not support offscreen bitmaps (negotiated in the Offscreen Bitmap Cache Capability Set described in [MS-RDPBCGR] section 2.2.7.1.10), or the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders, this order SHOULD be ignored.

3.2.5.1.3.1.3 Processing of Create NineGrid Bitmap

The structure and fields of the Create NineGrid Bitmap Alternate Secondary Drawing Order are specified in section 2.2.2.3.1.3.4. The order fields MUST be processed in accordance with this description.

If the client does not support rendering NineGrid bitmaps, negotiated in the DrawNineGrid Cache Capability Set (section 2.2.1.2), or the DrawNineGrid and MultiDrawNineGrid Primary Drawing Orders (see sections 3.2.5.1.1.1.21 and 3.2.5.1.1.1.22, respectively), this order SHOULD be ignored.

3.2.5.1.3.1.4 Processing of Stream Bitmap Orders

The structure and fields of the Stream Bitmap First and Stream Bitmap Next Alternate Secondary Drawing Orders are specified in sections 2.2.2.3.1.3.5.1 and 2.2.2.3.1.3.5.2, respectively. The order fields MUST be processed in accordance with this description.

The Stream Bitmap Orders are only used to populate the NineGrid Bitmap Cache (see section 3.1.1.1.6), and MUST follow immediately after the reception of a Create NineGrid Bitmap Alternate Secondary Drawing Order (see section 3.3.5.1.3.1.3). The NineGrid Bitmap Cache entry to populate with the streamed bitmap data is implicitly assumed to be the entry specified in the Create NineGrid Bitmap Order.

If the client does not support rendering NineGrid bitmaps, negotiated in the DrawNineGrid Cache Capability Set (section 2.2.1.2), or the DrawNineGrid or MultiDrawNineGrid Primary Drawing Orders (see sections 3.2.5.1.1.1.21 and 3.2.5.1.1.1.22, respectively), these orders SHOULD be ignored.

3.2.5.1.3.1.5 GDI+ Orders

3.2.5.1.3.1.5.1 Processing of Draw GDI+ Cache Orders

The structure and fields of the Draw GDI+ Cache First, Draw GDI+ Cache Next, and Draw GDI+ Cache End Alternate Secondary Drawing Orders are specified in sections 2.2.2.3.1.3.6.2, 2.2.2.3.1.3.6.3, and 2.2.2.3.1.3.6.4, respectively. The order fields MUST be processed in accordance with these descriptions.

If the client does not support rendering GDI+ 1.1 primitives (negotiated in the Draw GDI+ Capability Set Cache described in section 2.2.1.3), these orders SHOULD be ignored.

3.2.5.1.3.1.5.2 Processing of Draw GDI+ Orders

The structure and fields of the Draw GDI+ First, Draw GDI+ Next, and Draw GDI+ End Alternate Secondary Drawing Orders are specified in sections 2.2.2.3.1.3.6.5, 2.2.2.3.1.3.6.6, and 2.2.2.3.1.3.6.7, respectively. The order fields MUST be processed in accordance with these descriptions.

If the client does not support rendering GDI+ 1.1 primitives (negotiated in the Draw GDI+ Capability Set Cache described in section 2.2.1.3), these orders SHOULD be ignored.

3.2.5.2 Error Conditions

3.2.5.2.1 Sending of Bitmap Cache Error PDU

The Bitmap Cache Error PDU SHOULD be sent to a server that has requested the rendering of a cached bitmap that is not available in the client Bitmap Cache (see section 3.1.1.1.1).

Rendering of items from the bitmap cache is accomplished using the MemBlt (section 2.2.2.3.1.1.2.9) and Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Orders. If an error occurs during rendering, the client SHOULD complete the requested drawing operation by using a black bitmap in place of the requested cached bitmap, and then send this error PDU to the requesting server so that it can flush the appropriate caches and resend the drawing updates associated with the affected area.

The server will honor up to five Bitmap Cache Error PDUs for a given connection; further error PDUs are ignored to reduce server overhead.

The structure and fields of the Bitmap Cache Error PDU are specified in section 2.2.2.4.1.1, and the techniques described in [MS-RDPBCGR] section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.2.2 Sending of the Offscreen Bitmap Cache Error PDU

The Offscreen Bitmap Cache Error PDU SHOULD be transmitted when the creation of an offscreen bitmap in the Offscreen Bitmap Cache (see section 3.1.1.1.5) cannot be fulfilled due to a client-side failure (for example, low memory conditions). The Offscreen Bitmap Cache Error PDU SHOULD be sent to a server that requested the creation of the offscreen bitmap in the Offscreen Bitmap Cache (see section 3.1.1.1.5).

Creation of an offscreen bitmap is accomplished by using the Create Offscreen Bitmap (section 2.2.2.3.1.3.2) Alternate Secondary Drawing Order. The client SHOULD send this error PDU to the server so that it will resend the drawing updates associated with the affected area. On receiving this PDU, the server will disable offscreen bitmap caching for the duration of the connection; further errors should be ignored on the client side.

The structure and fields of the Offscreen Bitmap Cache Error PDU are specified in section 2.2.2.4.2, and the techniques described in [MS-RDPBCGR] section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.2.3 Sending of the DrawNineGrid Cache Error PDU

The DrawNineGrid Cache Error PDU SHOULD be transmitted when the creation of a NineGrid bitmap in the NineGrid Bitmap Cache (see section 3.1.1.1.6) cannot be fulfilled due to a client-side failure (for example, low memory conditions).

Creation of a NineGrid bitmap in the NineGrid Bitmap Cache is accomplished by first using the Create NineGrid Bitmap (section 2.2.2.3.1.3.4) Alternate Secondary Drawing Order, and then streaming the bitmap into the cache using the Stream Bitmap First (section 2.2.2.3.1.3.5.1) and Stream Bitmap Next (section 2.2.2.3.1.3.5.2) Alternate Secondary Drawing Orders. If processing of any of these orders fails, the client SHOULD send this error PDU to the server so that it will resend the drawing updates associated with the affected area. On receiving this PDU, the server will disable NineGrid bitmap caching for the duration of the connection; further errors should be ignored on the client side.

The structure and fields of the DrawNineGrid Cache Error PDU are specified in section 2.2.2.4.3, and the techniques described in [MS-RDPBCGR] section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.2.4 Sending of the GDI+ Error PDU

The GDI+ Error PDU SHOULD be transmitted when a GDI+ alternate secondary drawing order cannot be successfully processed due to a GDI+ rendering error or caching issue.

The client SHOULD send this PDU so that the server will resend GDI+ content as bitmaps without relying on the local client side GDI+ rendering. (The six GDI+ PDUs that are used to cache and render GDI+ primitives are specified in section 2.2.2.3.1.3.6.) On receiving this PDU, the server will also turn off GDI+ rendering for the duration of the connection; further errors should be ignored on the client side.

The structure and fields of the GDI+ Error PDU are specified in section 2.2.2.4.4, and the techniques described in [MS-RDPBCGR] section 3.2.5.1 demonstrate how to initialize the contents of the PDU. The contents of this PDU are not compressed.

3.2.5.3 Processing of the Server Redirection PDU

The principles behind server redirection and an example of how it operates within the context of an RDP connection is described in section 1.3.3.

Two variants of the Server Redirection PDU can be received by the client to indicate that it MUST terminate the current connection and reconnect to another server. The Standard Security variant (see section 2.2.3.2) of the Server Redirection PDU MUST be received when Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is not in effect. When Enhanced RDP Security is being used to secure the connection, the Enhanced Security variant (see section 2.2.3.3) of the PDU MUST be received.

The actual contents of the Server Redirection PDU (embedded in the Standard Security or Enhanced Security variant) are contained in a Server Redirection Packet (see section 2.2.3.1). The information required by the client to connect to a new target server MUST be specified in this PDU.

The techniques described in [MS-RDPBCGR] section 3.2.5.2 describe how to process the two variants of this PDU (the instructions regarding the Share Data Header MUST be ignored because it is not present in either PDU).

Once the client has completed processing the appropriate variant of this PDU, it MUST terminate the current connection to the server that transmitted the PDU and initiate a new connection to the target server specified in the Server Redirection Packet.

3.2.6 Timer Events

No client timer events are used.

3.2.7 Other Local Events

No additional events are used.

3.3 Server Details

3.3.1 Abstract Data Model

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

3.3.1.1 Persistent Bitmap Keys

The Persistent Bitmap Keys store holds a collection of 64-bit bitmap keys, each of which uniquely identifies a bitmap image that was sent to the client using a Cache Bitmap (Revision 2) Secondary Drawing Order (see section 2.2.2.3.1.2.6). When the server sends a bitmap to the client, it can first check the Persistent Bitmap Keys store to determine if the client already has the bitmap in a local bitmap cache (see section 3.1.1.1.1) and, hence, save on bandwidth.

3.3.1.2 Primary Drawing Order History

The Primary Drawing Order History store holds information on the fields that have been sent in primary drawing orders. There are three pieces of information that MUST be recorded:

Last primary order type constructed.

Current bounding rectangle.

Per-order record of the last value used in each field.

These records are updated as each primary drawing order is constructed, and are used to efficiently encode primary drawing orders for transmission on the wire using as few bytes as possible (see section 3.3.5.1.1.1).

3.3.1.3 Bitmap Cache Wait List

The Bitmap Cache Wait List stores a collection of 64-bit identifiers, each of which identifies a bitmap that is a candidate for placement in the Bitmap Cache (see section 3.1.1.1.1). The usage of the Bitmap Cache Wait List is negotiated using the Bitmap Cache (Revision 2) Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.2).

Bitmaps are placed into the Bitmap Cache Wait List the first time they are encountered in a server drawing operation. If any of the bitmaps in the Wait List are encountered during a subsequent drawing operation, they MUST be removed from the Wait List and placed into a Bitmap Cache.

Using a wait list in conjunction with persistent bitmap caching ensures that only bitmaps that are used more than once in rendering operations are written to persistent storage. In effect, it implies that a bitmap MUST be sent twice to the client before it is actually stored in a valid Bitmap Cache entry. (The first time it is sent to the client, it is used and not cached. The second time it is sent to the client, it is cached and then used.)

3.3.2 Timers

No server timers are used.

3.3.3 Initialization

No server initialization steps are specified.

3.3.4 Higher-Layer Triggered Events

No server higher-layer triggered events are used.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Drawing Orders

All drawing orders are encapsulated in an Orders Update (section 2.2.2.2), which is sent as part of the Graphics Update PDU (see [MS-RDPBCGR] section 2.2.9.1.1.3); or they are encapsulated in a Fast-Path Orders Update (section 2.2.2.3), which is sent as part of the Fast-Path Update PDU (see [MS-RDPBCGR] section 2.2.9.1.2).

There are three classes of drawing orders:

Primary Drawing Orders (section 3.2.5.1.1)

Secondary Drawing Orders (section 2.2.2.3.1.2)

Alternate Secondary Drawing Orders (section 2.2.2.3.1.3)

Orders belonging to each of these classes are packed together into an Orders Update structure or a Fast-Path Orders Update structure, each order being aligned on a byte boundary.

3.3.5.1.1 Primary Drawing Orders

3.3.5.1.1.1 Construction of a Primary Drawing Order

All primary drawing orders MUST conform to the structure and rules defined in section 2.2.2.3.1.1.2.

To efficiently construct a primary drawing order, the server MUST use a Primary Drawing Order History (section 3.2.1.1) store. This store holds three pieces of information:

- Last primary order type constructed.
- Current bounding rectangle.
- Per-order record of the last value used in each field.

These stored records allow the server to use the minimum amount of data when constructing an order; if a field is unchanged from the value that it had when the order type was last sent, it SHOULD NOT be included in the order being constructed. Hence, only fields that have new values are required to be sent to the client. The fields that are present in the order MUST be indicated by the fieldFlags field.

If all of the Coord-type fields (see section 2.2.2.3.1.1.1.1) in an order can be represented as a signed delta in the range -127 to 128 from the previous field value, the size of the order SHOULD be optimized by using delta-coordinates (see sections 2.2.2.3.1.1.1.1 and 2.2.2.3.1.1.2). In that case, all of the fields SHOULD be represented using delta-coordinates, and the TS_DELTA_COORDINATES (0x10) flag MUST be used in the primary drawing order header to indicate this fact.

Before a given order is sent, the server MUST also ensure that all of the data required to process the order is accessible to the client. For example, if the order refers to a cached item, that item MUST be present in the client-side cache when the order is processed. Or, if palletized color is being used, the correct palette MUST be applied at the client-side.

Once a primary drawing order has been constructed and transmitted to the client, the server MUST update the records in the Primary Drawing Order History (section 3.3.1.2) to ensure that future encodings use the minimum fields and data required.

3.3.5.1.1.1.1 Construction of DstBlt

The structure and fields of the DstBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.1. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The DstBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.2 Construction of MultiDstBlt

The structure and fields of the MultiDstBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.2. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.1.

The MultiDstBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.3 Construction of PatBlt

The structure and fields of the PatBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.3. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

If a cached brush is specified in this order, that brush MUST be sent to the client before this order is dispatched by using a Cache Brush Secondary Drawing Order (see section 3.3.5.1.2.1.6). (The client specifies support for brush caching using the Brush Capability Set defined in [MS-RDPBCGR] section 2.2.7.1.8.)

The PatBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.4 Construction of MultiPatBlt

The structure and fields of the MultiPatBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.4. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

If a cached brush is specified in this order, that brush MUST be sent to the client before this order is dispatched by using a Cache Brush Secondary Drawing Order (see section 3.3.5.1.2.1.6). (The client specifies support for brush caching using the Brush Capability Set defined in [MS-RDPBCGR] section 2.2.7.1.8.)

The MultiPatBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.5 Construction of OpaqueRect

The structure and fields of the OpaqueRect Primary Drawing Order are specified in section 2.2.2.3.1.1.2.5. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The OpaqueRect Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.6 Construction of MultiOpaqueRect

The structure and fields of the MultiOpaqueRect Primary Drawing Order are specified in section 2.2.2.3.1.1.2.6. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The MultiOpaqueRect Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.7 Construction of ScrBlt

The structure and fields of the ScrBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.7. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The ScrBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.8 Construction of MultiScrBlt

The structure and fields of the MultiScrBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.8. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The MultiScrBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.9 Construction of MemBlt

The structure and fields of the MemBlt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.9. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The source bitmap associated with the MemBlt Order MUST originate from one of three possible locations:

1. One of the negotiated bitmap caches (see section 3.1.1.1.1).
2. Offscreen Bitmap Cache (section 3.1.1.1.5).
3. Bitmap Cache Wait List (section 3.3.1.3).

If the source bitmap associated with the MemBlt Order refers to an item in one of the negotiated bitmap caches, the actual bitmap data MUST be sent to the client before this order is dispatched by using a Revision 1 or 2 Cache Bitmap Secondary Drawing Order (see sections 3.3.5.1.2.1.1 and 3.3.5.1.2.1.2).

If the source bitmap associated with the MemBlt Order refers to an item in the Offscreen Bitmap Cache, the actual entry MUST be created and initialized using the Create Offscreen Bitmap (see section 3.3.5.1.3.1.1) and Switch Surface (see section 3.3.5.1.3.1.2) Alternate Secondary Drawing Orders; the Create Offscreen Bitmap Order creates the offscreen bitmap while the Switch Surface Order is used to redirect all drawing operations to the offscreen bitmap. The cacheIndex field of the MemBlt Order MUST be set to the index of the Offscreen Bitmap Cache entry, and the bitmap cache ID (specified as part of the cacheId field) MUST be set to TS_BITMAPCACHE_SCREEN_ID (0xFF).

If the source bitmap associated with the MemBlt Order refers to a bitmap in the Bitmap Cache Wait List, the actual bitmap data MUST be sent to the client before this order is dispatched by using a Cache Bitmap (Revision 2) Secondary Drawing Order with the CBR2_DO_NOT_CACHE (0x10) flag set (see section 3.3.5.1.2.1.2). The cacheIndex field of the MemBlt Order MUST be set to BITMAPCACHE_WAITING_LIST_INDEX (32767), and the bitmap cache ID (specified as part of the cacheId field) MUST be set to the ID of the bitmap cache in which the bitmap will be stored when it is cached.

If palletized color is being used, the color table specified in the MemBlt Order MUST be sent to the client before this order is dispatched by using a Cache Color Table Secondary Drawing Order (see section 3.3.5.1.2.1.3). This step ensures that the client is able to render the cached bitmap correctly. Support for a Color Table Cache (see section 3.1.1.1.3) is implied by support for the MemBlt Order.

The MemBlt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.10 Construction of Mem3Blt

The structure and fields of the Mem3Blt Primary Drawing Order are specified in section 2.2.2.3.1.1.2.10. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The construction of the Mem3Blt Order follows the same principles as those outlined for the MemBlt Order specified in section 3.3.5.1.1.1.9. However, the Mem3Blt Order includes fields to specify a brush.

If a cached brush is specified in this order, that brush MUST be sent to the client before this order is dispatched by using a Cache Brush Secondary Drawing Order (see section 3.3.5.1.2.1.6). (The client specifies support for brush caching using the Brush Capability Set defined in [MS-RDPBCGR] section 2.2.7.1.8.)

The Mem3Blt Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.11 Construction of LineTo

The structure and fields of the LineTo Primary Drawing Order are specified in section 2.2.2.3.1.1.2.11. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The LineTo Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.12 Construction of SaveBitmap

The structure and fields of the SaveBitmap Primary Drawing Order are specified in section 2.2.2.3.1.1.2.12. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The SaveBitmap Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.13 Construction of GlyphIndex

The structure and fields of the GlyphIndex Primary Drawing Order are specified in section 2.2.2.3.1.2.5. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

All of the glyphs associated with the glyph cache indices specified in the order MUST be sent to the client before this order is dispatched by using a Revision 1 or 2 Cache Glyph Secondary Drawing Order (see sections 3.3.5.1.2.1.1 and 3.3.5.1.2.1.2) or a FastGlyph Primary Drawing Order (see section 3.3.5.1.1.1.15). The usage of glyph cache indices implies that support for glyph caching MUST have been negotiated by using the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9). If a fragment cache index is specified in a USE clause, that fragment MUST have been part of an ADD clause in a prior GlyphIndex or FastIndex order.

If a cached brush is specified in this order, that brush MUST be sent to the client before this order is dispatched by using a Cache Brush Secondary Drawing Order (see section 3.3.5.1.2.1.6). (The client specifies support for brush caching using the Brush Capability Set defined in [MS-RDPBCGR] section 2.2.7.1.8.)

The GlyphIndex Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.14 Construction of FastIndex

The structure and fields of the FastIndex Primary Drawing Order are specified in section 2.2.2.3.1.1.2.14. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The construction of the FastIndex Order follows the same principles as those outlined for the GlyphIndex Order specified in section 3.3.5.1.1.1.13. However, the FastIndex Order does not used cached brushes and also utilizes a more efficient field encoding.

The FastIndex Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.15 Construction of FastGlyph

The structure and fields of the FastGlyph Primary Drawing Order are specified in section 2.2.2.3.1.1.2.15. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

All of the glyphs associated with the glyph cache indices specified in the order MUST be sent to the client before this order is dispatched by using a Revision 1 or 2 Cache Glyph Secondary Drawing Order (see sections 2.2.2.3.1.2.2 and 2.2.2.3.1.2.3) or a prior FastGlyph Primary Drawing Order. The usage of glyph cache indices implies that support for glyph caching MUST have been negotiated by using the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9).

The FastGlyph Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.16 Construction of PolygonSC

The structure and fields of the PolygonSC Primary Drawing Order are specified in section 2.2.2.3.1.1.2.16. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The PolygonSC Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.17 Construction of PolygonCB

The structure and fields of the PolygonCB Primary Drawing Order are specified in section 2.2.2.3.1.1.2.17. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

If a cached brush is specified in this order, that brush MUST be sent to the client before this order is dispatched by using a Cache Brush Secondary Drawing Order (see section 3.3.5.1.2.1.6). (The client specifies support for brush caching using the Brush Capability Set defined in [MS-RDPBCGR] section 2.2.7.1.8.)

The PolygonCB Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.18 Construction of PolyLine

The structure and fields of the Polyline Primary Drawing Order are specified in section 2.2.2.3.1.1.2.18. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The Polyline Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.19 Construction of EllipseSC

The structure and fields of the EllipseSC Primary Drawing Order are specified in section 2.2.2.3.1.1.2.19. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The EllipseSC Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.20 Construction of EllipseCB

The structure and fields of the EllipseCB Primary Drawing Order are specified in section 2.2.2.3.1.1.2.20. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

If a cached brush is specified in this order, that brush MUST be sent to the client before this order is dispatched by using a Cache Brush Secondary Drawing Order (see section 3.3.5.1.2.1.6). (The client specifies support for brush caching using the Brush Capability Set defined in [MS-RDPBCGR] section 2.2.7.1.8.)

The EllipseCB Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.21 Construction of DrawNineGrid

The structure and fields of the DrawNineGrid Primary Drawing Order are specified in section 2.2.2.3.1.1.2.21. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

Support for the NineGrid Bitmap Cache (see section 3.1.1.1.6) MUST have been negotiated using the DrawNineGrid Cache Capability Set (section 2.2.1.2) because the order refers to the ID of a NineGrid-compatible bitmap in this cache. Furthermore, this bitmap MUST have been created and initialized before this order is dispatched by using the Create NineGrid Bitmap (see section 3.3.5.1.3.1.3) and Stream Bitmap (see section 3.3.5.1.3.1.4) Alternate Secondary Orders.

The DrawNineGrid Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.1.1.22 Construction of MultiDrawNineGrid

The structure and fields of the MultiDrawNineGrid Primary Drawing Order are specified in section 2.2.2.3.1.1.2.22. The order fields MUST be populated in accordance with this description and the instructions detailed in section 3.3.5.1.1.1.

The construction of the MultiDrawNineGrid Order follows the same principles as those outlined for the DrawNineGrid order specified in section 3.3.5.1.1.1.21. However, the MultiDrawNineGrid includes multiple clipping rectangles, as opposed to the DrawNineGrid Order, which only includes one clipping rectangle.

The MultiDrawNineGrid Order MUST NOT be sent to the client if support for it was not negotiated in the Order Capability Set (see [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.2 Secondary Drawing Orders

3.3.5.1.2.1 Construction of Secondary Drawing Orders

All secondary drawing orders are identified by the Secondary Drawing Order Header (section 2.2.2.3.1.2.1.1). Client support for a given secondary order is determined by the capabilities advertised by the client.

3.3.5.1.2.1.1 Construction of Cache Bitmap (Revision 1)

The structure and fields of the Cache Bitmap (Revision 1) Secondary Drawing Order are specified in section 2.2.2.3.1.2.2. The order fields MUST be populated in accordance with this description.

The bitmaps that are cached by the Cache Bitmap (Revision 1) Secondary Drawing Order can be consumed by future MemBlt (see section 3.3.5.1.1.1.9) and Mem3Blt (see section 3.3.5.1.1.1.10) Primary Drawing Orders.

The bitmap data MUST NOT include a Compressed Data Header structure (see [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.2) if the absence of this header was negotiated in the General Capability Set (see [MS-RDPBCGR] section 2.2.7.1.1).

The cache ID and cache index to use are selected by the server based on the management policies that are implemented (for example, an LRU policy might be in effect).

The Cache Bitmap (Revision 1) Order MUST NOT be sent to the client if support for bitmap caching was not negotiated using the Bitmap Cache (Revision 1) Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.1). Furthermore, if client-side support for the MemBlt or Mem3Blt Primary Drawing Orders (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3) does not exist, the Cache Bitmap (Revision 1) Order SHOULD NOT be sent to the client.

3.3.5.1.2.1.2 Construction of Cache Bitmap (Revision 2)

The structure and fields of the Cache Bitmap (Revision 2) Secondary Drawing Order are specified in section 2.2.2.3.1.2.3. The order fields MUST be populated in accordance with this description.

The bitmaps that are cached by the Cache Bitmap (Revision 2) Secondary Drawing Order can be consumed by future MemBlt (see section 3.3.5.1.1.1.9) and Mem3Blt (see section 3.3.5.1.1.1.11) Primary Drawing Orders.

The bitmap data MUST NOT include a Compressed Data Header structure (see [MS-RDPBCGR] section 2.2.9.1.1.3.1.2.2) if the absence of this header was negotiated in the General Capability Set (see [MS-RDPBCGR] section 2.2.7.1.1).

The cache ID and cache index to use are selected by the server based on the management policies that are implemented (for example, an LRU policy might be in effect).

If a 64-bit key identifying the bitmap MUST be generated for the bitmap (dictated by the capabilities for the bitmap cache negotiated in the Bitmap Cache (Revision 2) Capability Set as defined in [MS-RDPBCGR] section 2.2.7.1.5.2), the server MUST create a unique 64-bit identifier and include it in the Key1 and Key2 fields of the Cache Bitmap (Revision 2) Order.

If it is the first time the server is instructing the client to cache the bitmap, and a Bitmap Cache Wait List (see section 3.3.1.3) is present, the server MUST force the bitmap into the Wait List and set the CBR2_DO_NOT_CACHE (0x10) flag in the Cache Bitmap (Revision 2) Secondary Drawing Order. The cacheId field MUST be set to the ID of the bitmap cache in which the bitmap will be stored if it is encountered again. If the bitmap is encountered again, it MUST be removed from the Wait List and sent to the client without the CBR2_DO_NOT_CACHE flag.

The Cache Bitmap (Revision 2) Order MUST NOT be sent to the client if support for bitmap caching was not negotiated using the Bitmap Cache (Revision 2) Capability Set (see [MS-RDPBCGR] section 2.2.7.1.5.2). Furthermore, if client-side support for the MemBlt (see section 3.3.5.1.1.1.9) and Mem3Blt (see section 3.3.5.1.1.1.10) Primary Drawing Orders (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3) does not exist, the Cache Bitmap (Revision 2) Order SHOULD NOT be sent to the client.

3.3.5.1.2.1.3 Construction of Cache Color Table

The structure and fields of the Cache Color Table Secondary Drawing Order are specified in section 2.2.2.3.1.2.4. The order fields MUST be populated in accordance with this description.

The color tables that are cached by the Cache Color Table Secondary Drawing Order can be consumed by future MemBlt (see section 3.3.5.1.1.1.9) and Mem3Blt (see section 3.3.5.1.1.1.11) Primary Drawing Orders.

The cache ID and cache index to use are selected by the server based on the management policies that are implemented (for example, an LRU policy might be in effect).

The Cache Color Table Order SHOULD NOT be sent to the client if it does not support the MemBlt and Mem3Blt Primary Drawing Orders (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3).

3.3.5.1.2.1.4 Construction of Cache Glyph (Revision 1)

The structure and fields of the Cache Glyph (Revision 1) Secondary Drawing Order are specified in section 2.2.2.3.1.2.5. The order fields MUST be populated in accordance with this description.

The glyphs that are cached by the Cache Glyph (Revision 1) Secondary Drawing Order can be consumed by future GlyphIndex (see section 3.3.5.1.1.1.13), FastIndex (see section 3.3.5.1.1.1.14), and FastGlyph (see section 3.3.5.1.1.1.15) Primary Drawing Orders.

The cache index to use is selected by the server based on the management policies that are implemented (for example, an LRU policy might be in effect).

The Cache Glyph (Revision 1) Order MUST NOT be sent to the client if support for Revision 1 glyph caching was not negotiated using the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9). Furthermore, if client-side support for the GlyphIndex, FastIndex, or FastGlyph Primary Drawing Orders (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3) does not exist, the Cache Glyph (Revision 1) Order SHOULD NOT be sent to the client.

3.3.5.1.2.1.5 Construction of Cache Glyph (Revision 2)

The structure and fields of the Cache Glyph (Revision 2) Secondary Drawing Order are specified in section 2.2.2.3.1.2.6. The order fields MUST be populated in accordance with this description.

The construction of the Cache Glyph (Revision 2) Order follows the same principles as those outlined for the Cache Glyph (Revision 1) Order specified in section 3.3.5.1.2.1.4. However, the Cache Glyph (Revision 2) Order utilizes a more efficient field encoding.

The Cache Glyph (Revision 2) Order MUST NOT be sent to the client if support for Revision 2 glyph caching was not negotiated using the Glyph Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.9). Furthermore, if client-side support for the GlyphIndex, FastIndex, and FastGlyph Primary Drawing Orders (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3) does not exist, the Cache Glyph (Revision 2) Order SHOULD NOT be sent to the client.

3.3.5.1.2.1.6 Construction of Cache Brush

The structure and fields of the Cache Brush Secondary Drawing Order are specified in section 2.2.2.3.1.2.7. The order fields MUST be populated in accordance with this description.

The brushes that are cached by the Cache Brush Secondary Drawing Order can be consumed by a number of primary drawing orders:

- PatBlt (see section 3.3.5.1.1.1.3)
- MultiPatBlt (see section 3.3.5.1.1.1.4)
- Mem3Blt (see section 3.3.5.1.1.1.10)
- GlyphIndex (see section 3.3.5.1.1.1.13)
- PolygonCB (see section 3.3.5.1.1.1.17)
- EllipseCB (see section 3.3.5.1.1.1.20)

The cache index to use is selected by the server based on the management policies that are implemented (for example, an LRU policy might be in effect).

The Cache Brush Order MUST NOT be sent to the client if support for brush caching was not negotiated using the Brush Capability Set (see [MS-RDPBCGR] section 2.2.7.1.8). Furthermore, if the client does not support any primary drawing orders that use cached brushes, the Cache Brush Order SHOULD NOT be sent to the client.

3.3.5.1.3 Alternate Secondary Drawing Orders

3.3.5.1.3.1 Construction of Alternate Secondary Drawing Orders

All alternate secondary drawing orders MUST contain the Alternate Secondary Drawing Order Header (section 2.2.2.3.1.3.1.1). Client support for a given alternate secondary order is determined by the capabilities advertised by the client.

3.3.5.1.3.1.1 Construction of Create Offscreen Bitmap

The structure and fields of the Create Offscreen Bitmap Alternate Secondary Drawing Order are specified in section 2.2.2.3.1.3.2. The order fields MUST be populated in accordance with this description.

The offscreen bitmaps managed by the Create Offscreen Bitmap Alternate Secondary Drawing Order are specified in the Switch Surface Alternate Secondary Drawing Order (see section 3.3.5.1.3.1.2). By using the Switch Surface Order, the server can change the default client rendering surface to any one of the bitmaps created in the Offscreen Bitmap Cache by the Create Offscreen Order. Once drawing to an offscreen bitmap is complete, the server can direct the client to render the bitmap data to the primary drawing surface by using the MemBlt (see section 3.3.5.1.1.1.9) or Mem3Blt (see section 3.3.5.1.1.1.10) Primary Drawing Order.

The Create Offscreen Bitmap Order MUST NOT be sent to the client if support for offscreen bitmap caching was not negotiated using the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10), or if offscreen bitmap caching has been disabled due to the reception of an Offscreen Bitmap Cache Error PDU (see section 3.3.5.2.2). Furthermore, if client-side support for the MemBlt and Mem3Blt Primary Drawing Orders does not exist (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3), the Create Offscreen Bitmap Order SHOULD NOT be sent to the client.

3.3.5.1.3.1.2 Construction of Switch Surface

The structure and fields of the Switch Surface Alternate Secondary Drawing Order are specified in section 2.2.2.3.1.3.3. The order fields MUST be populated in accordance with this description.

The Create Offscreen Bitmap Alternate Secondary Drawing Order (see section 3.3.5.1.3.1.1), which is used to create the offscreen bitmap referenced in this order, MUST be sent to the client before this order is dispatched.

The Switch Surface Order MUST NOT be sent to the client if support for offscreen bitmap caching was not negotiated using the Offscreen Bitmap Cache Capability Set (see [MS-RDPBCGR] section 2.2.7.1.10), or if offscreen bitmap caching has been disabled due to the reception of an Offscreen Bitmap Cache Error PDU (see section 3.3.5.2.2). Furthermore, if client-side support for the MemBlt (see section 3.3.5.1.1.1.9) or Mem3Blt (see section 3.3.5.1.1.1.10) Primary Drawing Order does not exist (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3), the Switch Surface Order SHOULD NOT be sent to the client.

3.3.5.1.3.1.3 Construction of Create NineGrid Bitmap

The structure and fields of the Create NineGrid Alternate Secondary Drawing Order are specified in section 2.2.2.3.1.3.4. The order fields MUST be populated in accordance with this description.

The NineGrid bitmaps produced by the Create NineGrid Bitmap Alternate Secondary Drawing Order are initialized by the Stream Bitmap Alternate Secondary Drawing Orders (see section 3.3.5.1.3.1.4) and consumed by the DrawNineGrid and MultiDrawNineGrid Primary Drawing Orders (see sections 3.3.5.1.1.1.21 and 3.3.5.1.1.1.22).

The Create NineGrid Bitmap Order MUST NOT be sent to the client if support for NineGrid rendering was not negotiated using the DrawNineGrid Cache Capability Set (section 2.2.1.2), or if NineGrid bitmap caching has been disabled due to the reception of a DrawNineGrid Cache Error PDU (see section 3.3.5.2.3). Furthermore, if client-side support for the DrawNineGrid and MultiDrawNineGrid Primary Drawing Orders does not exist (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3), the Create NineGrid Bitmap Order SHOULD NOT be sent to the client.

3.3.5.1.3.1.4 Construction of Stream Bitmap Orders

The structure and fields of the Stream Bitmap First and Stream Bitmap Next Alternate Secondary Drawing Orders are specified in sections 2.2.2.3.1.3.5.1 and 2.2.2.3.1.3.5.2, respectively. The order fields MUST be populated in accordance with these descriptions.

The Stream Bitmap Orders are only used to populate the NineGrid Bitmap Cache (see section 3.1.1.1.6) and MUST follow immediately after the Create NineGrid Bitmap Alternate Secondary Drawing Order (see section 3.3.5.1.3.1.3). The NineGrid Bitmap Cache entry to populate with the streamed bitmap data is implicitly assumed to be the entry specified in the Create NineGrid Bitmap Order.

Because the Stream Bitmap Orders are only used to populate the NineGrid Bitmap Cache, they SHOULD NOT be sent to the client if support for NineGrid rendering was not negotiated using the DrawNineGrid Cache Capability Set (section 2.2.1.2), or if NineGrid bitmap caching has been disabled due to the reception of a DrawNineGrid Cache Error PDU (see section 3.3.5.2.3). Furthermore, if client-side support for the DrawNineGrid and MultiDrawNineGrid Primary Drawing Orders does not exist (negotiated using the Order Capability Set specified in [MS-RDPBCGR] section 2.2.7.1.3), the Stream Bitmap Orders SHOULD NOT be sent to the client.

3.3.5.1.3.1.5 GDI+ Orders

3.3.5.1.3.1.5.1 Construction of Draw GDI+ Cache Orders

The structure and fields of the Draw GDI+ Cache First, Draw GDI+ Cache Next, and Draw GDI+ Cache End Alternate Secondary Drawing Orders are specified in sections 2.2.2.3.1.3.6.2, 2.2.2.3.1.3.6.3, and 2.2.2.3.1.3.6.4, respectively. The order fields MUST be populated in accordance with these descriptions.

GDI+ 1.1 primitives cached by the Draw GDI+ Cache Orders are consumed by the Draw GDI+ Orders (see section 3.3.5.1.3.1.5.2).

The Draw GDI+ Cache Orders MUST NOT be sent to the client if support for GDI+ 1.1 rendering was not negotiated using the Draw GDI+ Capability Set (section 2.2.1.3), or if GDI+ 1.1 rendering has been disabled due to the reception of a GDI+ Error PDU (see section 3.3.5.2.4).

3.3.5.1.3.1.5.2 Construction of Draw GDI+ Orders

The structure and fields of the Draw GDI+ First, Draw GDI+ Next, and Draw GDI+ End Alternate Secondary Drawing Orders are specified in sections 2.2.2.3.1.3.6.5, 2.2.2.3.1.3.6.6, and 2.2.2.3.1.3.6.7, respectively. The order fields MUST be populated in accordance with these descriptions.

The Draw GDI+ Orders MUST NOT be sent to the client if support for GDI+ 1.1 rendering was not negotiated using the Draw GDI+ Capability Set (section 2.2.1.3), or if GDI+ 1.1 rendering has been disabled due to the reception of a GDI+ Error PDU (see section 3.3.5.2.4).

3.3.5.2 Error Conditions

3.3.5.2.1 Processing of Bitmap Cache Error PDU

The structure and fields of the Bitmap Cache Error PDU are specified in section 2.2.2.4.1, and the techniques described in [MS-RDPBCGR] section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the server MUST flush the appropriate Bitmap Cache entries (see section 3.1.1.1.1) and resend the graphics data associated with the affected area.

The server SHOULD only honor up to five Bitmap Cache Error PDUs for a given connection. If more than five PDUs have been sent, the server SHOULD ignore the PDU.

3.3.5.2.2 Processing of the Offscreen Bitmap Cache Error PDU

The structure and fields of the Offscreen Bitmap Cache Error PDU are specified in section 2.2.2.4.2, and the techniques described in [MS-RDPBCGR] section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the server MUST disable offscreen bitmap caching for the duration of the connection and resend the graphics data associated with the affected area.

Disabling offscreen caching implies that the server will not send the Create Offscreen Bitmap Alternate Secondary Drawing Order (see section 2.2.2.3.1.3.2).

3.3.5.2.3 Processing of the DrawNineGrid Cache Error PDU

The structure and fields of the DrawNineGrid Cache Error PDU are specified in section 2.2.2.4.3, and the techniques described in [MS-RDPBCGR] section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the server MUST disable NineGrid bitmap caching for the duration of the connection and resend the graphics data associated with the affected area.

Disabling NineGrid bitmap caching implies that the server will not send the Create NineGrid Bitmap (section 2.2.2.3.1.3.4) Alternate Secondary Drawing Order, and the Stream Bitmap First (section 2.2.2.3.1.3.5.1) and Stream Bitmap Next (section 2.2.2.3.1.3.5.2) Alternate Secondary Drawing Orders.

3.3.5.2.4 Processing of the GDI+ Error PDU

The structure and fields of the GDI+ Error PDU are specified in section 2.2.2.4.4, and the techniques described in [MS-RDPBCGR] section 3.3.5.2 demonstrate how to process the contents of the PDU.

Once this PDU has been processed, the server MUST disable GDI+ 1.1 rendering for the duration of the connection. All future GDI+ content MUST be sent as bitmaps so that local client-side GDI+ rendering is not required.

Disabling GDI+ 1.1 rendering implies that the six GDI+ PDUs that are used to cache and render GDI+ primitives (described in section 2.2.2.3.1.3.6) will not be sent.

3.3.5.3 Sending of the Server Redirection PDUs

The principles behind server redirection and an example of how it operates within the context of an RDP connection is described in section 2.2.3.

Two variants of the Server Redirection PDU are used to force the client to direct the current connection to another server. The Standard Security variant (see section 2.2.3.2) of the Server Redirection PDU MUST be used when Enhanced RDP Security (see [MS-RDPBCGR] section 5.4) is not in effect. When Enhanced RDP Security is being used to secure the connection, the Enhanced Security variant (see section 2.2.3.3) of the PDU MUST be used.

The actual contents of the Server Redirection PDU (embedded in the Standard Security or Enhanced Security variant) are contained in a Server Redirection Packet (see section 2.2.3.1). The server MUST initialize this structure with all of the information required by the client to connect to a new target server.

The techniques described in [MS-RDPBCGR] section 3.3.5.1 describe how to initialize the two variants of this PDU (the instructions regarding the Share Data Header MUST be ignored because it is not present in either PDU). The contents of this PDU are not compressed.

3.3.6 Timer Events

No server timer events are used.

3.3.7 Other Local Events

No additional events are used.

4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions.

4.1 Annotated Primary Drawing Orders

4.1.1 DstBlt

The following is an annotated dump of a DstBlt (section 2.2.2.3.1.1.2.1) Primary Drawing Order.

```
00000000 09 00 0c 48 00 37 01                              ...H.7.

09 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS_TYPE_CHANGE

00 -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_DSTBLT_ORDER

0c -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x0c
Binary of 0x0c = 0000 1100
Fields 3, 4 are present

DSTBLT_ORDER::nLeftRect not present
DSTBLT_ORDER::nTopRect not present

48 00 -> DSTBLT_ORDER::nWidth = 0x48 = 72
37 01 -> DSTBLT_ORDER::nHeight = 0x0137 = 311

DSTBLT_ORDER::bRop not present
```

4.1.2 MultiDstBlt

The following is an annotated dump of a MultiDstBlt (section 2.2.2.3.1.1.2.2) Primary Drawing Order.

```
00000000 09 0f 7f 12 01 2d 01 a0 00 0c 00 55 02 0d 00 04 .....-.....U....
00000010 81 12 81 2d 80 58 0c 80 60 80 40 0c             ...-.X..`.@.

09 -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS STANDARD |
  TS TYPE CHANGE

0f -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_MULTIDSTBLT_ORDER

7f -> PRIMARY DRAWING ORDER::fieldFlags = 0x7f
Binary of 0x7f = 0111 1111
Fields 1-7 are present

12 01 -> MULTI_DSTBLT_ORDER:nLeftRect = 0x0112 = 274
```

```
2d 01 -> MULTI DSTBLT ORDER:nTopRect = 0x012d = 301
a0 00 -> MULTI_DSTBLT_ORDER::nWidth = 0xa0 = 160
0c 00 -> MULTI_DSTBLT_ORDER::nHeight = 0x0c = 12

55 -> MULTI DSTBLT ORDER::bRop = 0x55 = ROP Table Entry #85 = 0x0009
02 -> MULTI DSTBLT ORDER::nDeltaEntries = 0x2 = 2 entries

0d 00 -> VARIABLE2_FIELD::cbData = 0xd = 13 bytes
04 81 12 81 2d 80 58 0c 80 60 80 40 0c -> VARIABLE2_FIELD::rgbData

04 -> DELTA RECTS FIELD::zeroBits = binary:0000 0100

Rectangle #1:
81 12 -> Delta Left = 0x112 = 274
81 2d -> Delta Top = 0x12d = 301
80 58 -> Delta Width = 0x58 = 88
0c -> Delta Height = 0xc = 12
Rectangle is (274, 301, 274 + 88 = 362, 301 + 12 = 313)

Rectangle #2:
80 60 -> Delta Left = 0x60 = 96
Delta Top = 0 (zeroBit is 1)
80 40 -> Delta Width = 0x40 = 64
0c -> Delta Height = 12
Rectangle is (274 + 96 = 370, 301 + 0 = 301, 370 + 64 = 434, 301 + 12 = 313)
```

4.1.3 PatBlt

The following is an annotated dump of a PatBlt (section 2.2.2.3.1.1.2.3) Primary Drawing Order.

```
00000000 09 01 7f 02 1a 00 c3 01 0d 00 0d 00 f0 ff ff 00 ................
00000010 5b ef 00 81                                     [...

09 -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS_TYPE_CHANGE

01 -> PRIMARY DRAWING ORDER::orderType = TS ENC PATBLT ORDER

7f 02 -> PRIMARY DRAWING ORDER::fieldFlags = 0x027f
Binary of 0x7f = 0000 0010 0111 1111
Fields 1-7, 10 are present

1a 00 -> PATBLT ORDER::nLeftRect = 0x1a = 26
c3 01 -> PATBLT ORDER::nTopRect = 0x01c3 = 451

0d 00 -> PATBLT_ORDER::nWidth = 0x0d = 13
0d 00 -> PATBLT_ORDER::nHeight = 0x0d = 13

f0 -> PATBLT ORDER::bRop = 0xf0 = ROP Table Entry #240 = 0x0021

ff ff 00 -> PATBLT ORDER::BackColor
TS_COLOR::RedOrPaletteIndex = 0xff
TS_COLOR::Green = 0xff
TS_COLOR::Blue = 0x00

5b ef 00 -> PATBLT ORDER::ForeColor
TS_COLOR::RedOrPaletteIndex = 0x5b
TS_COLOR::Green = 0xef
```

```
TS COLOR::Blue = 0x00

PATBLT_ORDER::BrushOrgX not present
PATBLT ORDER::BrushOrgY not present

81 --> PATBLT ORDER::BrushStyle = 0x81
0x81
= 0x01 |
  0x80
= BMF 1BPP |
  TS CACHED BRUSH

PATBLT ORDER::BrushHatch not present
PATBLT_ORDER::BrushExtra not present
```

4.1.4 MultiPatBlt

The following is an annotated dump of a MultiPatBlt (section 2.2.2.3.1.1.2.4) Primary Drawing Order.

```
00000000 09 10 df 31 e4 00 b7 00 0e 02 c5 01 5a ff ff 00 ...1........Z...
00000010 ae b2 04 19 00 08 40 80 e4 80 b7 82 0e 1b 1b 81 ......@.........
00000020 07 32 81 39 80 d5 32 fe c7 32 82 0e 81 78       .2.9..2..2...x

09 -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS STANDARD |
  TS TYPE CHANGE

10 -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_MULTIPATBLT_ORDER

df 31 -> PRIMARY DRAWING ORDER::fieldFlags = 0x31df
Binary of 0x31df = 0011 0001 1101 1111
Fields 1-5, 7-9, 13, 14 are present

e4 00 -> MULTI_PATBLT_ORDER::nLeftRect = 0xe4 = 228
b7 00 -> MULTI_PATBLT_ORDER::nTopRect = 0xb7 = 183

0e 02 -> MULTI PATBLT ORDER::nWidth = 0x020e = 526
c5 01 -> MULTI PATBLT ORDER::nHeight = 0x01c5 = 453

5a -> MULTI_PATBLT_ORDER::bRop = 0x5a = ROP Table Entry #90 = 0x0049

MULTI PATBLT ORDER::BackColor not present

ff ff 00 -> MULTI PATBLT ORDER::ForeColor
TS COLOR::RedOrPaletteIndex = 0xff
TS_COLOR::Green = 0xff
TS_COLOR::Blue = 0x00

ae -> MULTI PATBLT ORDER::BrushOrgX = 0xae = -82
b2 -> MULTI PATBLT ORDER::BrushOrgY = 0xb2 = -78

MULTI_PATBLT_ORDER:BrushStyle not present
MULTI_PATBLT_ORDER:BrushHatch not present
MULTI PATBLT ORDER:BrushExtra not present

04 -> MULTI PATBLT ORDER::nDeltaEntries = 0x04 = 4 entries

19 00 -> VARIABLE2_FIELD::cbData = 0x19 = 25 bytes
```

```
08 40 80 e4 80 b7 82 0e 1b 1b 81 07 32 81 39 80
d5 32 fe c7 32 82 0e 81 78 -> VARIABLE2_FIELD::rgbData

08 40 -> DELTA RECTS FIELD::zeroBits = binary:0000 1000 0100 0000

Rectangle #1:
80 e4 -> Delta Left = 0xe4 = 228
80 b7 -> Delta Top = 0xb7 = 183
82 0e -> Delta Width = 0x20e = 526
1b -> Delta Height = 0x1b = 27
Rectangle is (228, 183, 228 + 526 = 754, 183 + 27 = 210)

Rectangle #2:
Delta Left = 0 (zeroBit is 1)
1b -> Delta Top = 0x1b = 27
81 07 -> Delta Width = 0x107 = 263
32 -> Delta Height = 0x32 = 50
Rectangle is (228 + 0 = 228, 183 + 27 = 210, 228 + 263 = 491, 210 + 50 = 260)

Rectangle #3
81 39 -> Delta Left = 0x139 = 313
Delta Top = 0 (zeroBit is 1)
80 d5 -> Delta Width = 0xd5 = 213
32 -> Delta Height = 0x32 = 50
Rectangle is (228 + 313 = 541, 210 + 0 = 210, 541 + 213 = 754, 210 + 50 = 260)

Rectangle #4
fe c7 -> Delta Left = 0xfffffec7 = -313
32 -> Delta Top = 0x32 = 50
82 0e -> Delta Width = 0x20e = 526
81 78 -> Delta Height = 0x178 = 376
Rectangle is (541 - 313 = 228, 210 + 50 = 260, 228 + 526 = 754, 260 + 376 = 636)
```

4.1.5 OpaqueRect

The following is an annotated dump of an OpaqueRect (section 2.2.2.3.1.1.2.5) Primary Drawing Order.

```
00000000 09 0a 3c 00 04 00 03 73 02 06                   ..<....s..

09 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS_TYPE_CHANGE

0a -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_OPAQUERECT_ORDER

3c -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x3c
Binary of 0x3c = 0011 1100
Fields 3-6 are present

OPAQUERECT_ORDER::nLeftRect not present
OPAQUERECT_ORDER::nTopRect not present

00 04 -> OPAQUERECT_ORDER::nWidth = 0x0400 = 1024
00 03 -> OPAQUERECT_ORDER::nHeight = 0x0300 = 768
```

```
74 -> OPAQUERECT_ORDER::RedOrPaletteIndex = 0x74
02 -> OPAQUERECT_ORDER::Green = 0x02
06 -> OPAQUERECT_ORDER::Blue = 0x06
```

4.1.6 MultiOpaqueRect

The following is an annotated dump of a MultiOpaqueRect (section 2.2.2.3.1.1.2.6) Primary Drawing Order.

```
00000000 09 12 bf 01 87 01 1c 01 f1 00 12 00 5c ef 04 16 ............\...
00000010 00 08 40 81 87 81 1c 80 f1 01 01 01 10 80 f0 01 ..@.............
00000020 10 ff 10 10 80 f1 01                            .......

09 -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS_TYPE_CHANGE

12 -> PRIMARY DRAWING ORDER::orderType = TS ENC MULTIOPAQUERECT ORDER

bf 01 -> PRIMARY DRAWING ORDER::fieldFlags = 0x01bf
Binary of 0x01bf = 0000 0001 1011 1111
Fields 1-6,8-9 are present

87 01 -> MULTI OPAQUERECT ORDER::nLeftRect = 0x0187 = 391
1c 01 -> MULTI OPAQUERECT ORDER::nTopRect = 0x011c = 284

f1 00 -> MULTI OPAQUERECT ORDER::nWidth = 0x00f1 = 241
12 00 -> MULTI_OPAQUERECT_ORDER::nHeight = 0x0012 = 18

5c -> MULTI OPAQUERECT ORDER::RedOrPaletteIndex = 0x5c
ef -> MULTI OPAQUERECT ORDER::Green = 0xef
MULTI OPAQUERECT ORDER::Blue not present

04 -> MULTI_OPAQUERECT_ORDER::nDeltaEntries = 0x04 = 4 entries

16 00 -> VARIABLE2 FIELD::cbData = 0x16 = 22 bytes

08 40 81 87 81 1c 80 f1 01 01 01 10 80 f0 01 10
ff 10 10 80 f1 01 -> VARIABLE2_FIELD::rgbData

08 40 -> DELTA RECTS FIELD::zeroBits = binary:0000 1000 0100 0000

Rectangle #1:
81 87 -> Delta Left = 0x187 = 391
81 1c -> Delta Top = 0x11c = 284
80 f1 -> Delta Width = 0xf1 = 241
01 -> Delta Height = 0x1 = 1
Rectangle is (391, 284, 391 + 241 = 632, 284 + 1 = 285)

Rectangle #2:
Delta Left = 0 (zeroBit is 1)
01 -> Delta Top = 1
01 -> Delta Width = 1
10 -> Delta Height = 0x10 = 16
Rectangle is (391 + 0 = 391, 284 + 1 = 285, 391 + 1 = 392, 285 + 16 = 301)

Rectangle #3:
```

```
80 f0 -> Delta Left = 0xf0 = 240
Delta Top = 0 (zeroBit is 1)
01 -> Delta Width = 1
10 -> Delta Height = 0x10 = 16
Rectangle is (391 + 240 = 631, 285 + 0 = 285, 631 + 1 = 632, 285 + 16 = 301)

Rectangle #4:
ff 10 -> Delta Left = 0xffffff10 = -240
10 -> Delta Top = 0x10 = 16
80 f1 -> Delta Width = 0xf1 = 241
01 -> Delta Height = 0x1 = 1
Rectangle is (631 - 240 = 391, 285 + 16 = 301, 391 + 241 = 632, 301 + 1 = 302)
```

4.1.7 ScrBlt

The following is an annotated dump of an ScrBlt (section 2.2.2.3.1.1.2.7) Primary Drawing Order.

```
00000000 09 02 7d 07 00 a1 01 f1 00 cc 2f 01 8e          ..}........./..

09 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS_TYPE_CHANGE

02 -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_SCRBLT_ORDER

7d -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x7d
Binary of 0x7d = 0011 11101
Fields 1, 3-7 are present

07 00 -> SCRBLT_ORDER::nLeftRect = 0x07 = 7
SCRBLT_ORDER::nTopRect not present

a1 01 -> SCRBLT_ORDER::nWidth = 0x01a1 = 417
f1 00 -> SCRBLT_ORDER::nHeight  = 0x00f1 = 241

cc -> SCRBLT_ORDER::bRop = 0xcc = ROP Table Entry #204 = 0x0020

2f 01 -> SCRBLT_ORDER::nXSrc = 0x012f = 303
8e 00 -> SCRBLT_ORDER::nYSrc = 0x008e = 142
```

4.1.8 MultiScrBlt

The following is an annotated dump of a MultiScrBlt (section 2.2.2.3.1.1.2.8) Primary Drawing Order.

```
00000000 09 11 ff 01 2d 03 4e 01 10 00 10 00 cc 2d 03 05 ....-.N......-..
00000010 01 10 4a 00 00 00 00 00 00 00 00 00 83 2d 81 5d ..J..........-.]
00000020 10 01 01 7f 0f 01 01 7f 0e 01 01 7f 0d 01 01 7f ................
00000030 0c 01 01 7f 0b 01 01 7f 0a 01 01 7f 09 01 01 7f ................
00000040 08 01 01 7f 07 01 01 7f 06 01 01 7f 05 01 01 7f ................
00000050 04 01 01 7f 03 01 01 7f 02 01 01 7f 01 01       ..............
```

```
09 -> MULTI SCRBLT ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS STANDARD |
  TS TYPE CHANGE

11 -> MULTI_SCRBLT_ORDER::orderType = TS_ENC_MULTISCRBLT_ORDER

ff 01 -> MULTI SCRBLT ORDER::fieldFlags = 0x01ff
Binary of 0x01ff = 0000 0001 1111 1111
Fields 1-9 are present

2d 03 -> MULTI_SCRBLT_ORDER::nLeftRect = 0x032d = 813
4e 01 -> MULTI_SCRBLT_ORDER::nTopRect = 0x014e = 334

10 00 -> MULTI SCRBLT ORDER::nWidth = 0x10 = 16
10 00 -> MULTI SCRBLT ORDER::nHeight = 0x10 = 16

cc -> MULTI_SCRBLT_ORDER::bRop = 0xcc = ROP Table Entry #204 = 0x0020

2d 03 -> MULTI SCRBLT ORDER::nXSrc = 0x032d = 813
05 01 -> MULTI SCRBLT ORDER::nYSrc = 0x0105 = 261

10 -> MULTI_SCRBLT_ORDER::nDeltaEntries = 0x10 = 16 entries

4a 00 -> VARIABLE2_FIELD::cbData = 0x4a = 74 bytes

00 00 00 00 00 00 00 00 83 2d 81 5d 10 01 01 7f
0f 01 01 7f 0e 01 01 7f 0d 01 01 7f 0c 01 01 7f
0b 01 01 7f 0a 01 01 7f 09 01 01 7f 08 01 01 7f
07 01 01 7f 06 01 01 7f 05 01 01 7f 04 01 01 7f
03 01 01 7f 02 01 01 7f 01 01 -> VARIABLE2_FIELD::rgbData

00 00 00 00 00 00 00 00 -> DELTA RECTS FIELD::zeroBits = 0

Rectangle #1:
83 2d -> Delta Left -> 0x32d = 813
81 5d -> Delta Top -> 0x15d = 349
10 -> Delta Width -> 0x10 = 16
01 -> Delta Height -> 0x1 = 1
Rectangle is (813, 349, 813 + 16 = 829, 349 + 1 = 350)

Rectangle #2
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
0f -> Delta Width -> 0xf = 15
01 -> Delta Height -> 0x1 = 1
Rectangle is (813 + 1 = 814, 349 - 1 = 348, 814 + 15 = 829, 348 + 1 = 349)

Rectangle #3
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
0e -> Delta Width -> 0xe = 14
01 -> Delta Height -> 0x1 = 1
Rectangle is (815, 347, 829, 348)

Rectangle #4
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
0d -> Delta Width -> 0xd = 13
01 -> Delta Height -> 0x1 = 1
Rectangle is (816, 346, 829, 347)

Rectangle #5
```

```
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
0c -> Delta Width -> 0xc = 12
01 -> Delta Height -> 0x1 = 1
Rectangle is (817, 345, 829, 346)

Rectangle #6
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
0b -> Delta Width -> 0xb = 11
01 -> Delta Height -> 0x1 = 1
Rectangle is (818, 344, 829, 345)

Rectangle #7
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
0a -> Delta Width -> 0xa = 10
01 -> Delta Height -> 0x1 = 1
Rectangle is (819, 343, 829, 344)

Rectangle #8
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
09 -> Delta Width -> 0x9 = 9
01 -> Delta Height -> 0x1 = 1
Rectangle is (820, 342, 829, 343)

Rectangle #9
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
08 -> Delta Width -> 0x8 = 8
01 -> Delta Height -> 0x1 = 1
Rectangle is (821, 341, 829, 342)

Rectangle #10
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
07 -> Delta Width -> 0x7 = 7
01 -> Delta Height -> 0x1 = 1
Rectangle is (822, 340, 829, 341)

Rectangle #11
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
06 -> Delta Width -> 0x6 = 6
01 -> Delta Height -> 0x1 = 1
Rectangle is (823, 339, 829, 340)

Rectangle #12
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
05 -> Delta Width -> 0x5 = 5
01 -> Delta Height -> 0x1 = 1
Rectangle is (824, 338, 829, 339)

Rectangle #13
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
04 -> Delta Width -> 0x4 = 4
01 -> Delta Height -> 0x1 = 1
Rectangle is (825, 337, 829, 338)

Rectangle #14
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
```

```
03 -> Delta Width -> 0x3 = 3
01 -> Delta Height -> 0x1 = 1
Rectangle is (826, 336, 829, 337)

Rectangle #15
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
02 -> Delta Width -> 0x2 = 2
01 -> Delta Height -> 0x1 = 1
Rectangle is (827, 335, 829, 336)

Rectangle #16
01 -> Delta Left -> 0x1 = 1
7f -> Delta Top -> 0x7f = -1
01 -> Delta Width -> 0x1 = 1
01 -> Delta Height -> 0x1 = 1
Rectangle is (828, 334, 829, 335)
```

4.1.9 MemBlt

The following is an annotated dump of a MemBlt (section 2.2.2.3.1.1.2.9) Primary Drawing Order.

```
00000000 49 0d 97 ff 00 02 00 06 01 15 00 15 00          I............

49 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x19
0x49
= 0x01 |
  0x08 |
  0x40
= TS_STANDARD |
  TS_TYPE_CHANGE |
  TS_ZERO_FIELD_BYTE_BIT0

0d -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_MEMBLT_R2_ORDER

97 -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x97
TS_ZERO_FIELD_BYTE_BIT0 Indicates that last field byte is not present
PRIMARY_DRAWING_ORDER::fieldFlags = 0x0097
Binary of 0x0097 = 0000 0000 1001 0111
Fields 1-3, 5, 8 are present

ff 00 -> MEMBLT_ORDER::cacheId = 0x00ff
Color Table Cache Index = 0x00
Bitmap Cache ID = 0xff (TS_BITMAPCACHE_SCREEN_ID)

02 00 -> MEMBLT_ORDER::nLeftRect = 0x0002 = 2
06 01 -> MEMBLT_ORDER::nTopRect = 0x0106 = 262

MEMBLT_ORDER::nWidth not present
15 00 -> MEMBLT_ORDER::nHeight = 0x015 = 21

MEMBLT_ORDER::bRop not present

MEMBLT_ORDER::nXSrc not present
15 00 -> MEMBLT_ORDER::nYSrc = 0x015 = 21

MEMBLT_ORDER::cacheIndex not present
```

4.1.10 Mem3Blt

The following is an annotated dump of a Mem3Blt (section 2.2.2.3.1.1.2.10) Primary Drawing Order.

```
00000000 49 0e 3f 8e 01 00 3c 01 1f 01 40 00 0c 00 b8 ff I.?...<...@.....
00000010 ff 00 3b 1e ff 7f 03 71 00 21 04 05 40 0d 40 73 ..;....q.!..@.@s
00000020 07 20 1c 84 ff ff ff ff ff ff ff ff 00 21 fe 44 . ...........!.D
00000030 e7 38 81 c8 86 00 00 00 00 ff ff 00 00 ff ff ff .8..............
00000040 ff 43 c9 11 00 26 01 fd 43 f7 01 80 fd 5d 03 f0 .C...&..C....]..
00000050 01 00 00 81 00 00 00 00 09 00 c0 f3 00 c0 03 e0 ................
00000060 01 00 80 70 04 80 03 e0 08 00 c0 84 00 00 00 00 ...p............
00000070 ff ff ff ff 43 cf e0 79 01 26 40 20 3e 00 00 70 ....C..y.&@ >..p

49 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x19
0x49
= 0x01 |
  0x08 |
  0x40
= TS STANDARD |
  TS_TYPE_CHANGE |
  TS_ZERO_FIELD_BYTE_BIT0

0e -> PRIMARY DRAWING ORDER::orderType = TS ENC MEM3BLT R2 ORDER

3f 8e -> PRIMARY DRAWING ORDER::fieldFlags = 0x8e3f
TS ZERO FIELD BYTE BIT0 Indicates that last field byte is 0
PRIMARY_DRAWING_ORDER::fieldFlags = 0x008e3f
Binary of 0x008e3f = 0000 0000 1000 1110 0011 1111
Fields 1-6, 10-12, 16 are present

01 00 -> MEM3BLT ORDER::cacheId = 0x0001
Color Table Cache Index = 0x00
Bitmap Cache ID = 0x01

3c 01 -> MEM3BLT ORDER::nLeftRect = 0x013c = 316
1f 01 -> MEM3BLT ORDER::nTopRect = 0x011f = 287

40 00 -> MEM3BLT_ORDER::nWidth = 0x0040 = 64
0c 00 -> MEM3BLT_ORDER::nHeight = 0x015 = 12

b8 -> MEM3BLT ORDER::bRop = 0xb8

MEM3BLT ORDER::nXSrc not present
MEM3BLT_ORDER::nYSrc not present

MEM3BLT ORDER::BackColor not present

ff ff 00 -> MEM3BLT ORDER::ForeColor
TS COLOR::RedOrPaletteIndex = 0xff
TS_COLOR::Green = 0xff
TS_COLOR::Blue = 0x00

3b -> MEM3BLT ORDER::BrushOrgX = 0x3b = 59
1e -> MEM3BLT ORDER::BrushOrgY = 0x1e = 30

MEM3BLT_ORDER::BrushStyle not present
MEM3BLT_ORDER::BrushHatch not present
MEM3BLT ORDER::BrushExtra not present

ff 7f -> MEM3BLT_ORDER::cacheIndex = 0x7fff = 32767
```

4.1.11 LineTo

The following is an annotated dump of a LineTo (section 2.2.2.3.1.1.2.11) Primary Drawing Order.

```
00000000 1d 09 1e 02 a5 3d 03 1a 4a 03 1b 03 b1 0e a6 5b .....=..J.......[
00000010 ef 00                                           ..

1d -> PRIMARY_DRAWING_ORDER::controlFlags = 0x1d
0x1d
= 0x01 |
  0x04 |
  0x08 |
  0x10
= TS_STANDARD |
  TS_BOUNDS |
  TS TYPE CHANGE |
  TS DELTA COORDINATES

09 -> PRIMARY DRAWING ORDER::orderType = TS ENC LINETO ORDER

1e 02 -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x021e
Binary of 0x021e = 0000 0010 0001 1110

a5 -> PRIMARY DRAWING ORDER::bounds::flags = 0xa5
0xa5
= 0x01 |
  0x04 |
  0x20 |
  0x80
= TS BOUND LEFT |
  TS BOUND RIGHT |
  TS_BOUND_DELTA_TOP |
  TS_BOUND_DELTA_BOTTOM

3d 03 -> PRIMARY DRAWING ORDER::bounds::left = 0x033d = 829

1a -> PRIMARY DRAWING ORDER::bounds::top (delta) = 0x1a = 26 pixels from last top bounds
PRIMARY_DRAWING_ORDER::bounds::top = last bounds::top (0xe7) + 0x1a = 0x101 = 257

4a 03 -> PRIMARY DRAWING ORDER::bounds::right = 0x034a = 842

1b -> PRIMARY DRAWING ORDER::bounds::bottom (delta) = 0x1b = 27 pixels from last bottom
bounds
PRIMARY_DRAWING_ORDER::bounds::bottom = last bounds::bottom (0xf3) + 0x1b = 0x10e = 270

LINETO ORDER::BackMode not present

03 -> LINETO ORDER::nXStart(delta) = 0x03 = 3 pixels from last LINETO ORDER::nXStart
LINETO ORDER::nXStart = last LINETO ORDER::nXStart (0x33a) + 0x03 = 0x33d = 829

b1 -> LINETO_ORDER::nYStart(delta) = 0xb1 = -79 pixels from last LINETO_ORDER::nYStart
LINETO ORDER::nYStart = last LINETO ORDER::nYStart (0x15e) + 0xb1 = 0x10f = 271

0e -> LINETO ORDER::nXEnd(delta) = 0x0e = 14 pixels from last LINETO ORDER::nXEnd
LINETO ORDER::nXEnd = last LINETO ORDER::nXEnd (0x33d) + 0x0e = 0x34b = 843

a6 -> LINETO_ORDER::nYEnd(delta) = 0xa6 = -90 pixels from last LINETO_ORDER::nYEnd
LINETO ORDER::nYEnd = last LINETO ORDER::nYEnd (0x15b) + 0xa6 = 0x101 = 257

LINETO ORDER::BackColor not present
LINETO ORDER::bRop2 not present
LINETO_ORDER::PenStyle not present
LINETO_ORDER::PenWidth not present

5b ef 00 -> LINETO_ORDER::PenColor
```

```
TS COLOR::RedOrPaletteIndex = 0x5b
TS_COLOR::Green = 0xef
TS_COLOR::Blue = 0x00
```

4.1.12 SaveBitmap

The following is an annotated dump of a SaveBitmap (section 2.2.2.3.1.1.2.12) Primary Drawing Order.

```
00000000 19 0b 1b 74 45 00 00 79 70 10  ...tE..yp.

19 -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x19
= 0x01 |
  0x08 |
  0x10
= TS_STANDARD |
  TS TYPE CHANGE |
  TS DELTA COORDINATES

0b -> PRIMARY DRAWING ORDER::orderType = TS ENC SAVEBITMAP ORDER

1b -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x1b
Binary of 0x1e = 0001 1011
Fields 1-2, 4-5 are present

74 45 00 00 -> SAVEBITMAP_ORDER::SavedBitmapPosition = 0x4574 = 17780
Y granularity = 20
Y position = (17780 / (480 * 20)) * 20 = (17780 / 9600) * 20 = 1 * 20 = 20
X position = (17780 - (20 * 480)) / 20 = (17780 - 9600) / 20 = 8180 / 20 = 409

79 -> SAVEBITMAP ORDER::nLeftRect (delta) = 0x79 = 121
SAVEBITMAP_ORDER::nTopRect (delta) not present

70 -> SAVEBITMAP ORDER::nRightRect (delta) = 0x70 = 112
10 -> SAVEBITMAP ORDER::nBottomRect (delta) = 0x10 = 16

SAVEBITMAP_ORDER::Operation is not present
```

4.1.13 FastIndex

The following sections provide examples of annotated dumps of FastIndex (section 2.2.2.3.1.1.2.14) Primary Drawing Orders.

4.1.13.1 Example 1

First example of an annotated dump for FastIndex (section 2.2.2.3.1.1.2.14) Primary Drawing Order.

```
00000000 09 13 ff 70 07 00 03 ff ff 00 74 3b 00 0e 00 71 ...p......t;...q
00000010 00 42 00 7e 00 00 80 7c 00 15 00 00 01 06 02 04 .B.~...|........
00000020 03 08 05 09 06 06 06 06 07 06 08 02 ff 00 12    ................

99 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x99
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS_TYPE_CHANGE
```

```
13 -> PRIMARY_DRAWING_ORDER::orderType = 0x13 = TS_ENC_FAST_INDEX_ORDER

ff 70 -> PRIMARY DRAWING ORDER::fieldFlags = 0x70ff
Binary of 0x70ff = 0111 0000 1111 1111
Fields 1-8, 13-15 present

07 -> FASTINDEX_ORDER::cacheId = 0x07
00 03 -> FASTINDEX_ORDER::fDrawing = 0x0300
fAccel = 0x03
ulCharInc = 0x00

ff ff 00 -> FASTINDEX ORDER::BackColor
TS_COLOR::RedOrPaletteIndex = 0xff
TS_COLOR::Green = 0xff
TS COLOR::Blue = 0x00

74 3b 00 -> FASTINDEX ORDER::ForeColor
TS COLOR::RedOrPaletteIndex = 0x74
TS_COLOR::Green = 0x3b
TS COLOR::Blue = 0x00

0e 00 -> FASTINDEX ORDER::BkLeft = 0xe = 14
71 00 -> FASTINDEX ORDER::BkTop = 0x71 = 113
42 00 -> FASTINDEX_ORDER::BkRight = 0x42 = 66
7e 00 -> FASTINDEX_ORDER::BkBottom = 0x7e = 126

FASTINDEX ORDER::OpLeft not present
FASTINDEX ORDER::OpTop not present
FASTINDEX ORDER::OpRight not present
FASTINDEX ORDER::OpBottom not present

00 80 -> FASTINDEX_ORDER::X = 0x8000 = -32768
7c 00 -> FASTINDEX ORDER::Y = 0x7c = 124

15 -> VARIABLE1 FIELD::cbData = 0x15 = 21 bytes

00 00 01 06 02 04 03 08 05 09 06 06 06 06 07 06
08 02 ff 00 12 -> VARIABLE1_FIELD::rgbData

00 00 -> Glyph Cache Index = 0, Delta = 0
01 06 -> Glyph Cache Index = 1, Delta = 6
02 04 -> Glyph Cache Index = 2, Delta = 4
03 08 -> Glyph Cache Index = 3, Delta = 8
05 09 -> Glyph Cache Index = 4, Delta = 9
06 06 -> Glyph Cache Index = 5, Delta = 6
06 06 -> Glyph Cache Index = 6, Delta = 6
07 06 -> Glyph Cache Index = 7, Delta = 6
08 02 -> Glyph Cache Index = 8, Delta = 2

ff -> ADD Operation
00 -> Fragment Cache Index = 0
12 -> Size of Fragment = 0x12 = 18 bytes
```

4.1.13.2 Example 2

Second example of an annotated dump for FastIndex (section 2.2.2.3.1.1.2.14) Primary Drawing Order.

```
00000000 19 13 e0 60 e2 d5 e2 e2 03 fe 1c 00             ...`........

09 -> PRIMARY_DRAWING_ORDER::controlFlags = 0x09
0x19
```

```
= 0x01 |
  0x08 |
  0x10
= TS STANDARD |
  TS TYPE CHANGE |
  TS DELTA COORDINATES

13 -> PRIMARY_DRAWING_ORDER::orderType = 0x13 = TS_ENC_FAST_INDEX_ORDER

e0 60 -> PRIMARY DRAWING ORDER::fieldFlags = 0x60e0
Binary of 0x60e0 = 0110 0000 1110 0000
Fields 6-8, 14-15 are present

FASTINDEX_ORDER::cacheId not present
FASTINDEX_ORDER::fDrawing not present
FASTINDEX ORDER::BackColor not present
FASTINDEX ORDER::ForeColor not present

FASTINDEX ORDER::BkLeft not present

e2 -> FASTINDEX ORDER::BkTop(delta) = 0xe2 = -30 pixels from last FASTINDEX ORDER::BkTop
FASTINDEX ORDER::BkTop = last FASTINDEX ORDER::BkTop (0x159) + 0xe2 = 0x13b = 315

d5 -> FASTINDEX ORDER::BkRight(delta) = 0xd5 = -43 pixels from last
FASTINDEX_ORDER::BkRight
FASTINDEX_ORDER::BkRight = last FASTINDEX_ORDER::BkRight (0x161) + 0xd5 = 0x136 = 310

e2 -> FASTINDEX ORDER::BkBottom(delta) = 0xe2 = -30 pixels from last
FASTINDEX ORDER::BkBottom
FASTINDEX ORDER::BkBottom = last FASTINDEX ORDER::BkBottom (0x166) + 0xe2 = 0x148 = 328

FASTINDEX_ORDER::OpLeft not present
FASTINDEX_ORDER::OpTop not present
FASTINDEX ORDER::OpRight not present
FASTINDEX ORDER::OpBottom not present

FASTINDEX ORDER::X not present

e2 -> FASTINDEX_ORDER::Y(delta) = 0xe2 = -30 pixels from last FASTINDEX_ORDER::Y
FASTINDEX ORDER::Y = last FASTINDEX ORDER::Y(0x164) + 0xe2 = 0x146 = 326

03 -> VARIABLE1 FIELD::cbData = 0x03 = 3 bytes

fe 1c 00 -> VARIABLE1_FIELD::rgbData

fe -> USE Operation
1c -> Fragment Cache Index = 0x1c = 28
00 -> Delta
```

4.1.14 FastGlyph

The following is an annotated dump of a FastGlyph (section 2.2.2.3.1.1.2.15) Primary Drawing Order.

```
00000000 0d 18 fb 7e 0f 8b 00 b1 00 2c 04 bd 00 06 00 03 ...~.....,......
00000010 ff ff 00 8b 00 b1 00 93 00 be 00 0d 00 fe 7f 00 ................
00000020 80 00 80 bb 00 13 00 01 4a 06 0a 80 80 80 b8 c4 ........J.......
00000030 84 84 84 84 84 00 00 68 00                      .......h.

0d -> PRIMARY DRAWING ORDER::controlFlags
0x0d
= 0x80 | 0x04 | 0x01
= TS_TYPE_CHANGE | TS_BOUNDS | TS_STANDARD
```

```
18 -> PRIMARY_DRAWING_ORDER::orderorderType = TS_ENC_FAST_GLYPH_ORDER (0x18)

fb 7e -> PRIMARY DRAWING ORDER::fieldFlags = 0x7efb
Binary of 0x7efb = 0111 1110 1111 1011
Fields 1-2, 4-8, 10-15 are present

0f -> PRIMARY_DRAWING_ORDER::bounds (description flag)
0x0f
= 0x08 |
  0x04 |
  0x02 |
  0x01
= TS_BOUND_BOTTOM |
  TS_BOUND_RIGHT |
  TS BOUND TOP |
  TS BOUND LEFT

8b 00 -> PRIMARY DRAWING ORDER::bounds::left = 0x008b = 139
b1 00 -> PRIMARY_DRAWING_ORDER::bounds::top = 0x00b1 = 177
2c 04 -> PRIMARY DRAWING ORDER::bounds::right = 0x042c = 1068
bd 00 -> PRIMARY DRAWING ORDER::bounds::bottom = 0x00bd = 189

06 -> FASTGLYPH ORDER::cacheId = 6

00 03 -> FASTGLYPH_ORDER::fDrawing = 0x0300
flAccel = 0x03 = (SO_HORIZONTAL | SO_FLAG_DEFAULT_PLACEMENT)
ulCharInc = 0

FASTGLYPH ORDER::BackColor not present

ff ff 00 -> FASTGLYPH_ORDER::ForeColor
TS_COLOR::RedOrPaletteIndex = 0xff
TS COLOR::Green = 0xff
TS COLOR::Blue = 0x00

8b 00 -> FASTGLYPH ORDER::BkLeft = 139
b1 00 -> FASTGLYPH_ORDER::BkTop = 177
93 00 -> FASTGLYPH_ORDER::BkRight = 147
be 00 -> FASTGLYPH ORDER::BkBottom = 190

FASTGLYPH ORDER::OpLeft not present

0d 00 -> FASTGLYPH_ORDER::OpTop = 13
fe 7f -> FASTGLYPH ORDER::OpRight = 0x7ffe = 32766
00 80 -> FASTGLYPH ORDER::OpBottom = -32768

00 80 -> FASTGLYPH ORDER::X = -32768
bb 00 -> FASTGLYPH_ORDER::Y = 187

13 -> VARIABLE1 FIELD::cbData = 0x13 = 19 bytes

00 01 4a 06 0a 80 80 80 b8 c4 84 84 84 84 84 00
00 68 00 -> VARIABLE1 FIELD::rgbData

00 -> TS_CACHE_GLYPH_DATA_REV2::cacheIndex = 0
01 -> TS CACHE GLYPH DATA REV2::x = 1
4a -> TS CACHE GLYPH DATA REV2::y = -10
06 -> TS CACHE GLYPH DATA REV2::cx = 6
0a -> TS CACHE GLYPH DATA REV2::cy = 10

80 80 80 b8 c4 84 84 84 84 84 00 00 -> TS_CACHE_GLYPH_DATA_REV2::aj

0x80 -> X.....
0x80 -> X.....
```

```
0x80 -> X.....
0xb8 -> X XXX.
0xc4 -> XX...X
0x84 -> X....X
0x84 -> X....X
0x84 -> X....X
0x84 -> X....X
0x84 -> X....X

00 00 -> padding

68 00 -> Unicode = 0x68 = 104 = h
```

4.1.15 PolygonSC

The following is an annotated dump of a PolygonSC (section 2.2.2.3.1.1.2.16) Primary Drawing Order.

```
00000000 0d 14 7f 43 ae 00 b2 00 5c 36 02 0d 03 0d 01 20 ...C....\6.....
00000010 ba 00 36 d0 00 00 00 00 00 00 00 00 00 00 00 00 ..6.............
00000020 00 00 80 c0 76 80 b5 4a 80 9f ff a1 80 81 ff 7d ....v..J.......}
00000030 80 5d ff 5f 33 ff 4c 08 ff 3f 5c ff 3e ff b1 ff .]._3.L..?\.>...
00000040 46 ff 8a ff 59 ff 6b ff 73 ff 52 ff 96 ff 42 ff F...Y.k.s.R...B.
00000050 be ff 3e 69 ff 41 15 ff 51 80 40 ff 68 80 69 ff ..>i.A..Q.@.h.i.
00000060 87 80 8b ff ad 80 a7 58 80 b9 03 80 c2 2f 80 c1 .......X...../..
00000070 80 59 80 b5 80 7e 80 a1 80 9d 80 85 80 b1 80 60 .Y...~.........`
00000080 80 c0 38 80 c3 0c 80 bb 60 80 ab ff b5 80 91 ff ..8.....`.......
00000090 8e 80 70 ff 6e 80 49 ff 54 1e ff 44 71 ff 3d 46 ..p.n.I.T..Dq.=F
000000a0 ff 41 ff 9d ff 4f ff 7a ff 65 ff 5d ff 83 ff 49 .A...O.z.e.]...I
000000b0 ff aa ff 3f 53 ff 3e 7f ff 49 2b ff 5b 80 55 ff ...?S.>..I+.[.U.
000000c0 76 80 7b ff 9a 80 99 41 80 b1 6e 80 bf 19 80 c3 v.{....A..n.....
000000d0 80 45 80 bc 80 6c 80 ad 80 8e 80 94 80 a8 80 73 .E...l.........s
000000e0 81 7d 80 6f                                     .}.o

0d -> PRIMARY DRAWING ORDER::controlFlags = 0x0d
0x0d
= 0x01 |
  0X04 |
  0x08
= TS_STANDARD |
  TS BOUNDS |
  TS TYPE CHANGE

14 -> PRIMARY DRAWING ORDER::orderType = 0x14 = TS ENC POLYGON SC ORDER

7f -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x7f
Binary of 0x7f = 0111 1111
Fields 1-7 are present

43 -> PRIMARY DRAWING ORDER::bounds::flags = 0x43
0x43
= 0x01 |
  0x02 |
  0x40
= TS BOUND LEFT |
  TS BOUND TOP |
  TS_BOUND_DELTA_RIGHT

ae 00 -> PRIMARY DRAWING ORDER::bounds::left = 0x00ae = 174
b2 00 -> PRIMARY DRAWING ORDER::bounds::top = 0x00b2 = 178

5c -> PRIMARY_DRAWING_ORDER::bounds::right (delta) = 0x5c = 92 pixels from the last
bounds::right
```

```
PRIMARY DRAWING ORDER::bounds::right = last bounds::right (0x2f3) + 0x5c = 0x34f = 847

PRIMARY_DRAWING_ORDER::bounds::bottom not present

36 02 -> POLYGON SC ORDER::xStart = 0x0236 = 566
0d 03 -> POLYGON SC ORDER::yStart = 0x030d = 781

0d -> POLYGON_SC_ORDER::bRop2 = 0x0d = R2_COPYPEN
01 -> POLYGON_SC_ORDER::FillMode = 0x01 = ALTERNATE

20 ba 00 -> POLYGON SC ORDER::BrushColor
TS COLOR::RedOrPaletteIndex = 0x20
TS COLOR::Green = 0xba
TS_COLOR::Blue = 0x00

36 -> POLYGON SC ORDER::NumDeltaEntries = 0x36 = 54 entries

d0 -> VARIABLE1 FIELD::cbData = 0xd0 = 208 bytes

00 00 00 00 00 00 00 00 00 00 00 00 00 80 c0
76 80 b5 4a 80 9f ff a1 80 81 ff 7d 80 5d ff 5f
33 ff 4c 08 ff 3f 5c ff 3e ff b1 ff 46 ff 8a ff
59 ff 6b ff 73 ff 52 ff 96 ff 42 ff be ff 3e 69
ff 41 15 ff 51 80 40 ff 68 80 69 ff 87 80 8b ff
ad 80 a7 58 80 b9 03 80 c2 2f 80 c1 80 59 80 b5
80 7e 80 a1 80 9d 80 85 80 b1 80 60 80 c0 38 80
c3 0c 80 bb 60 80 ab ff b5 80 91 ff 8e 80 70 ff
6e 80 49 ff 54 1e ff 44 71 ff 3d 46 ff 41 ff 9d
ff 4f ff 7a ff 65 ff 5d ff 83 ff 49 ff aa ff 3f
53 ff 3e 7f ff 49 2b ff 5b 80 55 ff 76 80 7b ff
9a 80 99 41 80 b1 6e 80 bf 19 80 c3 80 45 80 bc
80 6c 80 ad 80 8e 80 94 80 a8 80 73 81 7d 80 6f -> VARIABLE1_FIELD::rgbData

00 00 00 00 00 00 00 00 00 00 00 00 00 00 -> DELTA PTS FIELD::zeroBits = 0

Point 1: 80 c0 -> Delta X = 0xc0 = 192            76 -> Delta Y = 0xfffffff6 = -10
Point 2: 80 b5 -> Delta X = 0xb5 = 181            4a -> Delta Y = 0xffffffca = -54
Point 3: 80 9f -> Delta X = 0x9f = 159            ff a1 -> Delta Y = 0xffffffa1 = -95
Point 4: 80 81 -> Delta X = 0x81 = 129            ff 7d -> Delta Y = 0xffffff7d = -131
Point 5: 80 5d -> Delta X = 0x5d = 93             ff 5f -> Delta Y = 0xffffff5f = -161
Point 6: 33 -> Delta X = 0x33 = 51                ff 4c -> Delta Y = 0xffffff4c = -180
Point 7: 08 -> Delta X = 0x8 = 8                  ff 3f -> Delta Y = 0xffffff3f = -193
Point 8: 5c -> Delta X = 0xffffffdc = -36         ff 3e -> Delta Y = 0xffffff3e = -194
Point 9: ff b1 -> Delta X = 0xffffffb1 = -79      ff 46 -> Delta Y = 0xffffff46 = -186
Point 10: ff 8a -> Delta X = 0xffffff8a = -118    ff 59 -> Delta Y = 0xffffff59 = -167
Point 11: ff 6b -> Delta X = 0xffffff6b = -149    ff 73 -> Delta Y = 0xffffff73 = -141
Point 12: ff 52 -> Delta X = 0xffffff52 = -174    ff 96 -> Delta Y = 0xffffff96 = -106
Point 13: ff 42 -> Delta X = 0xffffff42 = -190    ff be -> Delta Y = 0xffffffbe = -66
Point 14: ff 3e -> Delta X = 0xffffff3e = -194    69 -> Delta Y = 0xffffffe9 = -23
Point 15: ff 41 -> Delta X = 0xffffff41 = -191    15 -> Delta Y = 0x15 = 21
Point 16: ff 51 -> Delta X = 0xffffff51 = -175    80 40 -> Delta Y = 0x40 = 64
Point 17: ff 68 -> Delta X = 0xffffff68 = -152    80 69 -> Delta Y = 0x69 = 105
Point 18: ff 87 -> Delta X = 0xffffff87 = -121    80 8b -> Delta Y = 0x8b = 139
Point 19: ff ad -> Delta X = 0xffffffad = -83     80 a7 -> Delta Y = 0xa7 = 167
Point 20: 58 -> Delta X = 0xffffffd8 = -40        80 b9 -> Delta Y = 0xb9 = 185
Point 21: 03 -> Delta X = 0x3 = 3                 80 c2 -> Delta Y = 0xc2 = 194
Point 22: 2f -> Delta X = 0x2f = 47               80 c1 -> Delta Y = 0xc1 = 193
Point 23: 80 59 -> Delta X = 0x59 = 89            80 b5 -> Delta Y = 0xb5 = 181
Point 24: 80 7e -> Delta X = 0x7e = 126           80 a1 -> Delta Y = 0xa1 = 161
Point 25: 80 9d -> Delta X = 0x9d = 157           80 85 -> Delta Y = 0x85 = 133
Point 26: 80 b1 -> Delta X = 0xb1 = 177           80 60 -> Delta Y = 0x60 = 96
Point 27: 80 c0 -> Delta X = 0xc0 = 192           38 -> Delta Y = 0x38 = 56
Point 28: 80 c3 -> Delta X = 0xc3 = 195           0c -> Delta Y = 0xc = 12
Point 29: 80 bb -> Delta X = 0xbb = 187           60 -> Delta Y = 0xffffffe0 = -32
Point 30: 80 ab -> Delta X = 0xab = 171           ff b5 -> Delta Y = 0xffffffb5 = -75
```

```
Point 31: 80 91 -> Delta X = 0x91 = 145          ff 8e -> Delta Y = 0xffffff8e = -114
Point 32: 80 70 -> Delta X = 0x70 = 112          ff 6e -> Delta Y = 0xffffff6e = -146
Point 33: 80 49 -> Delta X = 0x49 = 73           ff 54 -> Delta Y = 0xffffff54 = -172
Point 34: 1e -> Delta X = 0x1e = 30              ff 44 -> Delta Y = 0xffffff44 = -188
Point 35: 71 -> Delta X = 0xfffffff1 = -15       ff 3d -> Delta Y = 0xffffff3d = -195
Point 36: 46 -> Delta X = 0xffffffc6 = -58       ff 41 -> Delta Y = 0xffffff41 = -191
Point 37: ff 9d -> Delta X = 0xffffff9d = -99    ff 4f -> Delta Y = 0xffffff4f = -177
Point 38: ff 7a -> Delta X = 0xffffff7a = -134   ff 65 -> Delta Y = 0xffffff65 = -155
Point 39: ff 5d -> Delta X = 0xffffff5d = -163   ff 83 -> Delta Y = 0xffffff83 = -125
Point 40: ff 49 -> Delta X = 0xffffff49 = -183   ff aa -> Delta Y = 0xffffffaa = -86
Point 41: ff 3f -> Delta X = 0xffffff3f = -193   53 -> Delta Y = 0xffffffd3 = -45
Point 42: ff 3e -> Delta X = 0xffffff3e = -194   7f -> Delta Y = 0xffffffff = -1
Point 43: ff 49 -> Delta X = 0xffffff49 = -183   2b -> Delta Y = 0x2b = 43
Point 44: ff 5b -> Delta X = 0xffffff5b = -165   80 55 -> Delta Y = 0x55 = 85
Point 45: ff 76 -> Delta X = 0xffffff76 = -138   80 7b -> Delta Y = 0x7b = 123
Point 46: ff 9a -> Delta X = 0xffffff9a = -102   80 99 -> Delta Y = 0x99 = 153
Point 47: 41 -> Delta X = 0xffffffc1 = -63       80 b1 -> Delta Y = 0xb1 = 177
Point 48: 6e -> Delta X = 0xffffffee = -18       80 bf -> Delta Y = 0xbf = 191
Point 49: 19 -> Delta X = 0x19 = 25              80 c3 -> Delta Y = 0xc3 = 195
Point 50: 80 45 -> Delta X = 0x45 = 69           80 bc -> Delta Y = 0xbc = 188
Point 51: 80 6c -> Delta X = 0x6c = 108          80 ad -> Delta Y = 0xad = 173
Point 52: 80 8e -> Delta X = 0x8e = 142          80 94 -> Delta Y = 0x94 = 148
Point 53: 80 a8 -> Delta X = 0xa8 = 168          80 73 -> Delta Y = 0x73 = 115
Point 54: 81 7d -> Delta X = 0x17d = 381         80 6f -> Delta Y = 0x6f = 111

Actual point 1: 758, 771
Actual point 2: 939, 717
Actual point 3: 1098, 622
Actual point 4: 1227, 491
Actual point 5: 1320, 330
Actual point 6: 1371, 150
Actual point 7: 1379, -43
Actual point 8: 1343, -237
Actual point 9: 1264, -423
Actual point 10: 1146, -590
Actual point 11: 997, -731
Actual point 12: 823, -837
Actual point 13: 633, -903
Actual point 14: 439, -926
Actual point 15: 248, -905
Actual point 16: 73, -841
Actual point 17: -79, -736
Actual point 18: -200, -597
Actual point 19: -283, -430
Actual point 20: -323, -245
Actual point 21: -320, -51
Actual point 22: -273, 142
Actual point 23: -184, 323
Actual point 24: -58, 484
Actual point 25: 99, 617
Actual point 26: 276, 713
Actual point 27: 468, 769
Actual point 28: 663, 781
Actual point 29: 850, 749
Actual point 30: 1021, 674
Actual point 31: 1166, 560
Actual point 32: 1278, 414
Actual point 33: 1351, 242
Actual point 34: 1381, 54
Actual point 35: 1366, -141
Actual point 36: 1308, -332
Actual point 37: 1209, -509
Actual point 38: 1075, -664
Actual point 39: 912, -789
Actual point 40: 729, -875
```

```
Actual point 41: 536, -920
Actual point 42: 342, -921
Actual point 43: 159, -878
Actual point 44: -6, -793
Actual point 45: -144, -670
Actual point 46: -246, -517
Actual point 47: -309, -340
Actual point 48: -327, -149
Actual point 49: -302, 46
Actual point 50: -233, 234
Actual point 51: -125, 407
Actual point 52: 17, 555
Actual point 53: 185, 670
Actual point 54: 566, 781
```

4.1.16 PolygonCB

The following is an annotated dump of a PolygonCB (section 2.2.2.3.1.1.2.17) Primary Drawing Order.

```
00000000 09 15 ef 1b ea 00 46 01 0d 01 08 00 00 04 03 81 ......F.........
00000010 08 03 05 88 09 26 09 77                         .....&.w

09 -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x09
= 0x01 |
  0x08
= TS_STANDARD |
  TS TYPE CHANGE

15 -> PRIMARY DRAWING ORDER::orderType = 0x15 = TS ENC POLYGON CB ORDER

ef 1b -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x1bef
Binary of 0x1bff = 0001 1011 1110 1111
Fields 1-4, 6-10, 12-13 are present

ea 00 -> POLYGON CB ORDER::xStart = 0xea = 234
46 01 -> POLYGON CB ORDER::yStart = 0x146 = 326

0d -> POLYGON_CB_ORDER::bRop2 = 0x0d = COPYPEN
01 -> POLYGON CB ORDER::FillMode = 0x01 = ALTERNATE

POLYGON CB ORDER::BackColor not present

08 00 00 -> POLYGON_CB_ORDER::ForeColor
TS_COLOR::RedOrPaletteIndex = 0x08
TS COLOR::Green = 0x00
TS COLOR::Blue = 0x00

04 -> POLYGON CB ORDER::BrushOrgX = 4
03 -> POLYGON_CB_ORDER::BrushOrgY = 3

81 -> POLYGON CB ORDER:BrushStyle = 0x81 = 0x80 | 0x01 = TS CACHED BRUSH | BMF 1BPP
08 -> POLYGON CB ORDER:BrushHatch = Brush Cache Index = 0x8

POLYGON CB ORDER::BrushExtra not present

03 -> POLYGON_CB_ORDER::NumDeltaEntries = 0x3 = 3 entries

05 -> VARIABLE1 FIELD::cbData = 5 bytes

88 09 26 09 77 -> VARIABLE1_FIELD::rgbData
```

```
88 -> DELTA PTS FIELD::zeroBits = binary:10 00 10 00

Point 1: Delta X = 0 (zeroBit set)      09 -> Delta Y = 0x9 = 9
Point 2: 26 -> Delta X = 0x26 = 38      09 -> Delta Y = 0x9 = 9
Point 3: Delta X = 0 (zeroBit set)      77 -> Delta Y = 0xfffffff7 = -9

Actual point 1: 234, 335
Actual point 2: 272, 344
Actual point 3: 272, 335
```

4.1.17 Polyline

The following is an annotated dump of a Polyline (section 2.2.2.3.1.1.2.18) Primary Drawing Order.

```
00000000 2d 16 73 f8 01 b8 02 00 c0 00 20 6c 00 00 00 00 -.s....... l....
00000010 00 04 00 00 ff 7e 76 ff 41 6c ff 24 62 ff 2b 59 .....~v.Al.$b.+Y
00000020 ff 55 51 ff 9c 49 73 43 80 4d ff be 80 99 ff ba .UQ..IsC.M......
00000030 80 cd ff b7 80 de ff b6 80 ca ff b6 80 96 ff b7 ................
00000040 80 48 ff ba 6f ff be ff 97 43 ff 52 4a ff 2b 51 .H..o....C.RJ.+Q
00000050 ff 24 59 ff 44 63 ff 81 6c 56 76 2f 80 82 0a 80 .$Y.Dc..lVv/....
00000060 bf 14 80 dd 1e 80 d4 27 80 ab 2f 80 64 37 0d 3d .......'../.d7.=
00000070 ff b3 80 42 ff 67 80 46                         ...B.g.F

2d -> PRIMARY DRAWING ORDER::controlFlags = 0x2d
0x2d
= 0x01 |
  0X04 |
  0x08 |
  0x20
= TS STANDARD |
  TS BOUNDS |
  TS TYPE CHANGE |
  TS_ZERO_BOUNDS_DELTAS

16 -> PRIMARY DRAWING ORDER::orderType = 0x16 = TS ENC POLYLINE ORDER

73 -> PRIMARY DRAWING ORDER::fieldFlags = 0x73
Binary of 0x73 = 1110011
Fields 1-2, 5-7 are present

PRIMARY DRAWING ORDER::bounds = last bounds

f8 01 -> POLYLINE ORDER::xStart = 0x01f8 = 504
b8 02 -> POLYLINE ORDER::yStart = 0x02b8 = 696

POLYLINE_ORDER::bRop2 not present
POLYLINE ORDER::BrushCacheEntry not present

00 c0 00 -> POLYLINE ORDER::PenColor::Red = 0x00
TS COLOR::RedOrPaletteIndex = 0x00
TS_COLOR::Green = 0xc0
TS_COLOR::Blue = 0x00

20 -> POLYLINE ORDER::NumDeltaEntries = 0x20 = 32 entries

6c -> VARIABLE1 FIELD::cbData = 0x6c = 108 bytes

00 00 00 00 00 04 00 00 ff 7e 76 ff 41 6c ff 24
62 ff 2b 59 ff 55 51 ff 9c 49 73 43 80 4d ff be
80 99 ff ba 80 cd ff b7 80 de ff b6 80 ca ff b6
80 96 ff b7 80 48 ff ba 6f ff be ff 97 43 ff 52
4a ff 2b 51 ff 24 59 ff 44 63 ff 81 6c 56 76 2f
80 82 0a 80 bf 14 80 dd 1e 80 d4 27 80 ab 2f 80
```

```
64 37 0d 3d ff b3 80 42 ff 67 80 46 -> VARIABLE1 FIELD::rgbData

00 00 00 00 00 04 00 00 -> DELTA_PTS_FIELD::zeroBits

Point 1: ff 7e -> Delta X = 0xffffff7e = -130      76 -> Delta Y = 0xfffffff6 = -10
Point 2: ff 41 -> Delta X = 0xffffff41 = -191      6c -> Delta Y = 0xfffffec = -20
Point 3: ff 24 -> Delta X = 0xffffff24 = -220      62 -> Delta Y = 0xffffffe2 = -30
Point 4: ff 2b -> Delta X = 0xffffff2b = -213      59 -> Delta Y = 0xffffffd9 = -39
Point 5: ff 55 -> Delta X = 0xffffff55 = -171      51 -> Delta Y = 0xffffffd1 = -47
Point 6: ff 9c -> Delta X = 0xffffff9c = -100      49 -> Delta Y = 0xffffffc9 = -55
Point 7: 73 -> Delta X = 0xfffffff3 = -13          43 -> Delta Y = 0xffffffc3 = -61
Point 8: 80 4d -> Delta X = 0x4d = 77              ff be -> Delta Y = 0xffffffbe = -66
Point 9: 80 99 -> Delta X = 0x99 = 153             ff ba -> Delta Y = 0xffffffba = -70
Point 10: 80 cd -> Delta X = 0xcd = 205            ff b7 -> Delta Y = 0xffffffb7 = -73
Point 11: 80 de -> Delta X = 0xde = 222            ff b6 -> Delta Y = 0xffffffb6 = -74
Point 12: 80 ca -> Delta X = 0xca = 202            ff b6 -> Delta Y = 0xffffffb6 = -74
Point 13: 80 96 -> Delta X = 0x96 = 150            ff b7 -> Delta Y = 0xffffffb7 = -73
Point 14: 80 48 -> Delta X = 0x48 = 72             ff ba -> Delta Y = 0xffffffba = -70
Point 15: 6f -> Delta X = 0xffffffef = -17         ff be -> Delta Y = 0xffffffbe = -66
Point 16: ff 97 -> Delta X = 0xffffff97 = -105     43 -> Delta Y = 0xffffffc3 = -61
Point 17: ff 52 -> Delta X = 0xffffff52 = -174     4a -> Delta Y = 0xffffffca = -54
Point 18: ff 2b -> Delta X = 0xffffff2b = -213     51 -> Delta Y = 0xffffffd1 = -47
Point 19: ff 24 -> Delta X = 0xffffff24 = -220     59 -> Delta Y = 0xffffffd9 = -39
Point 20: ff 44 -> Delta X = 0xffffff44 = -188     63 -> Delta Y = 0xffffffe3 = -29
Point 21: ff 81 -> Delta X = 0xffffff81 = -127     6c -> Delta Y = 0xffffffec = -20
Point 22: 56 -> Delta X = 0xffffffd6 = -42         76 -> Delta Y = 0xfffffff6 = -10
Point 23: 2f -> Delta X = 0x2f = 47                Delta Y = 0 (due to zeroBit being set)
Point 24: 80 82 -> Delta X = 0x82 = 130            0a -> Delta Y = 0xa = 10
Point 25: 80 bf -> Delta X = 0xbf = 191            14 -> Delta Y = 0x14 = 20
Point 26: 80 dd -> Delta X = 0xdd = 221            1e -> Delta Y = 0x1e = 30
Point 27: 80 d4 -> Delta X = 0xd4 = 212            27 -> Delta Y = 0x27 = 39
Point 28: 80 ab -> Delta X = 0xab = 171            2f -> Delta Y = 0x2f = 47
Point 29: 80 64 -> Delta X = 0x64 = 100            37 -> Delta Y = 0x37 = 55
Point 30: 0d -> Delta X = 0xd = 13                 3d -> Delta Y = 0x3d = 61
Point 31: ff b3 -> Delta X = 0xffffffb3 = -77      80 42 -> Delta Y = 0x42 = 66
Point 32: ff 67 -> Delta X = 0xffffff67 = -153     80 46 -> Delta Y = 0x46 = 70

Actual point 1: 374, 686
Actual point 2: 183, 666
Actual point 3: -37, 636
Actual point 4: -250, 597
Actual point 5: -421, 550
Actual point 6: -521, 495
Actual point 7: -534, 434
Actual point 8: -457, 368
Actual point 9: -304, 298
Actual point 10: -99, 225
Actual point 11: 123, 151
Actual point 12: 325, 77
Actual point 13: 475, 4
Actual point 14: 547, -66
Actual point 15: 530, -132
Actual point 16: 425, -193
Actual point 17: 251, -247
Actual point 18: 38, -294
Actual point 19: -182, -333
Actual point 20: -370, -362
Actual point 21: -497, -382
Actual point 22: -539, -392
Actual point 23: -492, -392
Actual point 24: -362, -382
Actual point 25: -171, -362
Actual point 26: 50, -332
Actual point 27: 262, -293
Actual point 28: 433, -246
```

```
Actual point 29: 533, -191
Actual point 30: 546, -130
Actual point 31: 469, -64
Actual point 32: 316, 6
```

4.1.18 EllipseSC

The following is an annotated dump of an EllipseSC (section 2.2.2.3.1.1.2.19) Primary Drawing Order.

```
00000000 0d 19 7f 29 ca 00 fb 3c 02 ca 00 47 01 eb 03 3c ...)...<...G...<
00000010 02 0d 02 c0 21 00                               ....!.

0d -> PRIMARY DRAWING ORDER::controlFlags = 0x09
0x0d
= 0x01 |
  0x04 |
  0x08
= TS STANDARD |
  TS BOUNDS |
  TS TYPE CHANGE

19 -> PRIMARY_DRAWING_ORDER::orderType = TS_ENC_ELLIPSE_SC_ORDER

7f -> PRIMARY DRAWING ORDER::fieldFlags = 0x7f
Binary of 0x7f = 0111 1111
Fields 1-7 are present

29 -> PRIMARY DRAWING ORDER::bounds::flags = 0x29
0x29
= 0x01 |
  0x08 |
  0x20
= TS_BOUND_LEFT |
  TS BOUND BOTTOM |
  TS BOUND DELTA TOP

ca 00 -> PRIMARY DRAWING ORDER::bounds::left = 0xca = 202

fb -> PRIMARY_DRAWING_ORDER::bounds::top(delta) = 0xfb = -5 pixels from last bounds::top
PRIMARY DRAWING ORDER::bounds::top = last bounds::top (0x14c) + 0xfb = 0x147 = 327

PRIMARY DRAWING ORDER::bounds::right not present

3c 02 -> PRIMARY_DRAWING_ORDER::bounds::bottom = 0x023c = 572

ca 00 -> ELLIPSE SC ORDER::LeftRect = 0x00ca = 202
47 01 -> ELLIPSE SC ORDER::TopRect = 0x0147 = 327

eb 03 -> ELLIPSE SC ORDER::RightRect = 0x03eb = 1003
3c 02 -> ELLIPSE_SC_ORDER::BottomRect = 0x023c = 572

0d -> ELLIPSE SC ORDER::bRop2 = 0x0d = COPYPEN
02 -> ELLIPSE SC ORDER::FillMode = 0x02 = WINDING

c0 21 00 -> ELLIPSE SC ORDER::Color
TS_COLOR::RedOrPaletteIndex = 0xc0
TS_COLOR::Green = 0x21
TS_COLOR::Blue = 0x00
```

4.1.19 EllipseCB

The following is an annotated dump of an EllipseCB (section 2.2.2.3.1.1.2.20) Primary Drawing Order.

```
00000000 0d 1a ff 0c b0 03 b9 c9 09 00 8e fe 10 01 5e 02 ..............^.
00000010 0d 01 ff ff 00 60 30 00 02 05                   .....`0...

0d -> PRIMARY_DRAWING_ORDER::controlFlags = 0x09
0x0d
= 0x01 |
  0x04 |
  0x08
= TS_STANDARD |
  TS_BOUNDS |
  TS_TYPE_CHANGE

1a -> PRIMARY DRAWING ORDER::orderType = TS ENC ELLIPSE CB ORDER

ff 0c -> PRIMARY DRAWING ORDER::fieldFlags = 0x0cff
Binary of 0x0cff = 0000 1100 1111 1111
Fields 1-8, 11-12 are present

b0 -> PRIMARY DRAWING ORDER::bounds::flags = 0x29
0xb0
= 0x10 |
  0x20 |
  0x80
= TS BOUND DELTA LEFT |
  TS BOUND DELTA TOP |
  TS BOUND DELTA BOTTOM

03 -> PRIMARY_DRAWING_ORDER::bounds::left(delta) = 0x03 = 3 pixels from last bounds::left
PRIMARY DRAWING ORDER::bounds::left = last bounds::left (0x2e) + 0x03 = 0x32 = 49

b9 -> PRIMARY DRAWING ORDER::bounds::top(delta) = 0xb9 = -71 pixels from last bounds::top
PRIMARY DRAWING ORDER::bounds::top = last bounds::top (0x89) + 0xb9 = 0x42 = 66

PRIMARY_DRAWING_ORDER::bounds::right not present

c9 -> PRIMARY DRAWING ORDER::bounds::bottom(delta) = 0xc9 = -55 pixels from last
bounds::bottom
PRIMARY DRAWING ORDER::bounds::bottom = last bounds::bottom (0xad) + 0xc9 = 0x76 = 118

09 00 -> ELLIPSE_CB_ORDER::LeftRect = 0x0009 = 9
8e fe -> ELLIPSE CB ORDER::TopRect = 0xfe8e = -370

10 01 -> ELLIPSE CB ORDER::RighttRect = 0x0110 = 272
5e 02 -> ELLIPSE CB ORDER::BottomRect = 0x025e = 606

0d -> ELLIPSE_CB_ORDER::bRop2 = 0x0d = COPYPEN
01 -> ELLIPSE CB ORDER::FillMode = 0x01 = ALTERNATE

ff ff 00 -> ELLIPSE CB ORDER::BackColor
TS COLOR::RedOrPaletteIndex = 0xff
TS_COLOR::Green = 0xff
TS_COLOR::Blue = 0x00

60 30 00 -> ELLIPSE CB ORDER::ForeColor::Red = 0xff
TS COLOR::RedOrPaletteIndex = 0x60
TS COLOR::Green = 0x30
TS_COLOR::Blue = 0x00

ELLIPSE CB ORDER::BrushOrgX not present
ELLIPSE_CB_ORDER::BrushOrgY not present
```

```
02 -> ELLIPSE_CB_ORDER::BrushStyle = 0x02 = BMF_4BPP
05 -> ELLIPSE_CB_ORDER::BrushHatch = Brush Cache Index = 0x05

ELLIPSE_CB_ORDER::BrushExtra not present
```

4.1.20 DrawNineGrid

The following is an annotated dump of a DrawNineGrid (section 2.2.2.3.1.1.2.21) Primary Drawing Order.

```
00000000 1d 07 1c 87 39 03 86 00 4e 03 7d fb f9 0d       ....9...N.}...

1d -> PRIMARY DRAWING ORDER::controlFlags = 0x1d
0x1d
= 0x01 |
  0X04 |
  0x08 |
  0x10
= TS STANDARD |
  TS BOUNDS |
  TS_TYPE_CHANGE |
  TS_DELTA_COORDINATES

07 -> PRIMARY DRAWING ORDER::orderType = 0x07 = TS ENC DRAWNINEGRID ORDER

1c -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x1c
Binary of 0x1c = 0001 1100
Fields 3-5 are present

87 -> PRIMARY DRAWING ORDER::bounds::flags = 0x87
0x87
= 0x01 |
  0x02 |
  0x04 |
  0x80
= TS BOUND LEFT |
  TS BOUND TOP |
  TS_BOUND_RIGHT |
  TS_BOUND_DELTA_BOTTOM

39 03 -> PRIMARY DRAWING ORDER::bounds::left = 0x0393 = 925
86 00 -> PRIMARY DRAWING ORDER::bounds::top = 0x86 = 134

4e 03 -> PRIMARY_DRAWING_ORDER::bounds::right = 0x034e = 846
7d -> PRIMARY_DRAWING_ORDER::bounds::bottom(delta) = 0x7d = -131 pixels from the last
bounds::bottom
PRIMARY DRAWING ORDER::bounds::bottom = last bounds::bottom (0x11e) + 0x7d = 0x9b = 155

DRAWNINEGRID ORDER Order::srcLeft not present
DRAWNINEGRID_ORDER_Order::srcTop not present

fb -> DRAWNINEGRID ORDER Order::srcRight (delta) = 0xfb = -5 pixels from last srcRight
DRAWNINEGRID ORDER Order::srcRight = last DRAWNINEGRID ORDER Order::srcRight (0x26) +
0xfb = 0x21

f9 -> DRAWNINEGRID_ORDER_Order::srcBottom (delta) = 0xf9 = -7 pixels from last srcBottom
DRAWNINEGRID_ORDER_Order::srcBottom = last DRAWNINEGRID_ORDER_Order::srcBottom (0x28) +
0xf9 = 0x21

0d -> DRAWNINEGRID_ORDER_Order::bitmapId = 13
```

4.1.21 MultiDrawNineGrid

The following is an annotated dump of a MultiDrawNineGrid (section 2.2.2.3.1.1.2.22) Primary Drawing Order.

```
00000000 1d 08 7c 03 9b 00 7c fe 03 02 a4 00 01 05 00 40  ..|...|........@
00000010 80 9b 05 1a                                      ....

1d -> PRIMARY_DRAWING_ORDER::controlFlags = 0x1d
0x1d
= 0x01 |
  0X04 |
  0x08 |
  0x10
= TS_STANDARD |
  TS_BOUNDS |
  TS TYPE CHANGE |
  TS DELTA COORDINATES

08 -> PRIMARY DRAWING ORDER::orderType = 0x08 = TS ENC MULTI DRAWNINEGRID ORDER

7c -> PRIMARY_DRAWING_ORDER::fieldFlags = 0x7c
Binary of 0x7c = 1111100
Fields 3-7 are present

03 -> PRIMARY_DRAWING_ORDER::bounds::flags = 0x3
0x3
= 0x1 |
  0x2
= TS BOUND LEFT |
  TS BOUND TOP

9b 00 -> PRIMARY_DRAWING_ORDER::bounds::left = 0x9b = 155
7c fe -> PRIMARY DRAWING ORDER::bounds::top = 0xfec7 = -388
PRIMARY DRAWING ORDER::bounds::right not present
PRIMARY DRAWING ORDER::bounds::bottom not present

MULTI_DRAWNINEGRID_ORDER_Order::srcLeft not present
MULTI_DRAWNINEGRID_ORDER_Order::srcTop not present

03 -> MULTI DRAWNINEGRID ORDER Order::srcRight (delta) = 0x3 = 3 pixels from last
srcRight
MULTI DRAWNINEGRID ORDER Order::srcRight = last MULTI DRAWNINEGRID ORDER Order::srcRight
(0x02) + 0x3 = 5

02 -> MULTI DRAWNINEGRID ORDER Order::srcBottom (delta) = 0x2 = 2 pixels from last
srcBottom
MULTI DRAWNINEGRID ORDER Order::srcBottom = last
MULTI DRAWNINEGRID ORDER Order::srcBottom (0x01) + 0x2 = 3

a4 00 -> MULTI_DRAWNINEGRID_ORDER_Order::bitmapId = 0xa4

01 -> MULTI DRAWNINEGRID ORDER Order::nDeltaEntries = 0x01 = 1 entry

05 00 -> VARIABLE2 FIELD::cbData = 0x0005 = 5 bytes

40 80 9b 05 1a-> VARIABLE2_FIELD::rgbData

40 -> DELTA RECTS FIELD::zeroBits = binary:0100 0000

Rectangle #1:
80 9b -> Delta Left = 0x9b = 155
Delta Top = 0 (zeroBit is set)
05 -> Delta Width = 0x5 = 5
1a -> Delta Height = 0x1a = 26
```

```
Rectangle is (155, 0, 155 + 5 = 160, 0 + 26 = 26)
```

4.2 Annotated Secondary Drawing Orders

4.2.1 Cache Bitmap (Revision 2)

The following is an annotated dump of a Cache Bitmap (Revision 2) (section 2.2.2.3.1.2.3) Secondary Drawing Order.

```
00000000 03 da 00 a1 0c 05 20 40 dc ff ff 85 ff ff 99 d6 ...... @........
00000010 99 d6 99 d6 99 d6 06 8b 99 d6 99 d6 99 d6 10 84 ................
00000020 08 42 08 42 10 84 99 d6 99 d6 99 d6 99 d6 06 84 .B.B............
00000030 99 d6 99 d6 99 d6 ff ff 16 69 99 d6 06 69 99 d6 .........i...i..
00000040 04 cc 89 52 03 6e ff ff 02 6e 08 42 01 70 08 42 ...R.n...n.B.p.B
00000050 71 ff ff ce 18 c6 01 81 08 42 ce 66 29 02 cd 89 q........B.f)...
00000060 52 03 88 10 84 99 d6 99 d6 99 d6 00 00 00 00 00 R...............
00000070 00 00 00 d8 99 d6 03 f8 01 00 00 00 00 f0 66 99 ..............f.
00000080 d6 05 6a 99 d6 00 c4 cc 89 52 03 6e ff ff 02 6e ..j......R.n...n
00000090 08 42 01 70 08 42 71 ff ff ce 18 c6 01 81 08 42 .B.p.Bq........B
000000a0 ce 66 29 02 cd 89 52 03 00 04 d6 99 d6 c3 80 61 .f)...R........a
000000b0 00 a5 80 40 ec 52 00 5a 00 2d 00 24 00 12 00 24 ...@.R.Z.-.$...$
000000c0 00 12 00 5a 00 2d 00 a5 80 52 00 c3 80 61 00 00 ...Z.-...R...a..
000000d0 00 00 00 cc 89 52 03 6e ff ff 02 cb 18 c6 84 08 .....R.n........
000000e0 42 08 42 08 42 ff ff                           B.B.B..

03 -> SECONDARY DRAWING ORDER HEADER::controlFlags = 0x03
0x03
= 0x2 |
  0x1
= TS SECONDARY |
  TS STANDARD

da 00 -> SECONDARY_DRAWING_ORDER_HEADER::orderLength = 0xda + 13 = 231 bytes

a1 0c -> CACHE BITMAP REV2 ORDER::header::extraFlags = 0x0ca1 = binary:000011001 0100 001
cacheId = binary:001 = 2
bitsPerPixelID = binary:0100 = 0x4 (16 bpp)

flags = binary:11001 = 0x19
0x19
= 0x10 |
  0x08 |
  0x01
= CBR2_DO_NOT_CACHE |
  CBR2_NO_BITMAP_COMPRESSION_HEADER |
  CBR2 HEIGHT SAME AS WIDTH

05 -> CACHE BITMAP REV2 ORDER::orderType = 0x05 = TS CACHE BITMAP COMPRESSED REV2

20 -> CACHE BITMAP REV2 ORDER::bitmapWidth = 0x20 = 32
CACHE_BITMAP_REV2_ORDER::bitmapHeight = 32 (CBR2_HEIGHT_SAME_AS_WIDTH flag is set)

40 dc -> CACHE BITMAP REV2 ORDER::bitmapLength = 0xdc = 220 bytes
ff ff -> CACHE BITMAP REV2 ORDER::cacheIndex = 0x7fff = 32767

Compressed bitmap data (220 bytes):
00000000 85 ff ff 99 d6 99 d6 99 d6 99 d6 06 8b 99 d6 99 ................
00000010 d6 99 d6 10 84 08 42 08 42 10 84 99 d6 99 d6 99 ......B.B.......
00000020 d6 99 d6 06 84 99 d6 99 d6 99 d6 ff ff 16 69 99 ..............i.
00000030 d6 06 69 99 d6 04 cc 89 52 03 6e ff ff 02 6e 08 ..i.....R.n...n.
00000040 42 01 70 08 42 71 ff ff ce 18 c6 01 81 08 42 ce B.p.Bq........B.
00000050 66 29 02 cd 89 52 03 88 10 84 99 d6 99 d6 99 d6 f)...R..........
```

```
00000060 00 00 00 00 00 00 00 00 d8 99 d6 03 f8 01 00 00 ................
00000070 00 00 f0 66 99 d6 05 6a 99 d6 00 c4 cc 89 52 03 ...f...j......R.
00000080 6e ff ff 02 6e 08 42 01 70 08 42 71 ff ff ce 18 n...n.B.p.Bq....
00000090 c6 01 81 08 42 ce 66 29 02 cd 89 52 03 00 04 d6 ....B.f)...R....
000000a0 99 d6 c3 80 61 00 a5 80 40 ec 52 00 5a 00 2d 00 ....a...@.R.Z.-.
000000b0 24 00 12 00 24 00 12 00 5a 00 2d 00 a5 80 52 00 $...$...Z.-...R.
000000c0 c3 80 61 00 00 00 00 00 cc 89 52 03 6e ff ff 02 ..a.......R.n...
000000d0 cb 18 c6 84 08 42 08 42 08 42 ff ff             .....B.B.B..

Decompressed bitmap data:
00000000 ff ff 99 d6 99 d6 99 d6 99 d6 00 00 00 00 00 00 ................
00000010 00 00 00 00 00 00 99 d6 99 d6 99 d6 10 84 08 42 ...............B
00000020 08 42 10 84 99 d6 99 d6 99 d6 99 d6 00 00 00 00 .B..............
00000030 00 00 00 00 00 00 00 00 99 d6 99 d6 99 d6 ff ff ................
00000040 ff ff 99 d6 99 d6 99 d6 99 d6 00 00 00 00 00 00 ................
00000050 00 00 00 00 00 00 99 d6 99 d6 99 d6 10 84 08 42 ...............B
00000060 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000070 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000080 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000090 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
000000a0 08 42 10 84 10 84 10 84 10 84 10 84 10 84 10 84 .B..............
000000b0 10 84 10 84 10 84 10 84 10 84 10 84 99 d6 ff ff ................
000000c0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
000000d0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff 08 42 ...............B
000000e0 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
000000f0 08 42 08 42 08 42 08 42 08 42 08 42 08 42 ff ff .B.B.B.B.B.B..
00000100 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
00000110 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
00000120 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
00000130 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
00000140 ff ff 10 84 10 84 10 84 10 84 10 84 10 84 10 84 ................
00000150 10 84 10 84 10 84 10 84 10 84 10 84 10 84 08 42 ...............B
00000160 08 42 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000170 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000180 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000190 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
000001a0 08 42 10 84 99 d6 99 d6 99 d6 00 00 00 00 00 00 .B..............
000001b0 00 00 00 00 00 00 99 d6 99 d6 99 d6 99 d6 ff ff ................
000001c0 ff ff 99 d6 99 d6 99 d6 00 00 00 00 00 00 00 00 ................
000001d0 00 00 00 00 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
000001e0 08 42 10 84 99 d6 99 d6 99 d6 00 00 00 00 00 00 .B..............
000001f0 00 00 00 00 00 00 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000200 ff ff 99 d6 99 d6 99 d6 00 00 00 00 00 00 00 00 ................
00000210 00 00 00 00 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
00000220 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000230 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000240 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000250 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
00000260 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000270 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000280 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000290 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
000002a0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
000002b0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
000002c0 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
000002d0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
000002e0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
000002f0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000300 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000310 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
00000320 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000330 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ................
00000340 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000350 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
00000360 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
```

```
00000370 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000380 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ..............
00000390 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
000003a0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
000003b0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
000003c0 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ..............
000003d0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
000003e0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
000003f0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000400 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ..............
00000410 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
00000420 08 42 10 84 10 84 10 84 10 84 10 84 10 84 10 84 .B..............
00000430 10 84 10 84 10 84 10 84 10 84 10 84 99 d6 ff ff ..............
00000440 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ..............
00000450 ff ff ff ff ff ff ff ff ff ff ff ff ff ff 08 42 ..............B
00000460 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
00000470 08 42 08 42 08 42 08 42 08 42 08 42 08 42 ff ff .B.B.B.B.B.B..
00000480 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
00000490 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
000004a0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ..............
000004b0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ..............
000004c0 ff ff 10 84 10 84 10 84 10 84 10 84 10 84 10 84 ..............
000004d0 10 84 10 84 10 84 10 84 10 84 10 84 10 84 08 42 ..............B
000004e0 08 42 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
000004f0 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000500 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ..............
00000510 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
00000520 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000530 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000540 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ..............
00000550 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
00000560 08 42 10 84 99 d6 99 d6 99 d6 00 00 00 00 99 d6 .B..............
00000570 99 d6 99 d6 99 d6 00 00 00 00 99 d6 99 d6 ff ff ..............
00000580 ff ff 99 d6 99 d6 99 d6 00 00 00 00 99 d6 99 d6 ..............
00000590 99 d6 99 d6 00 00 00 00 99 d6 99 d6 10 84 08 42 ..............B
000005a0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 00 00 00 00 .B..............
000005b0 99 d6 99 d6 00 00 00 00 99 d6 99 d6 99 d6 ff ff ..............
000005c0 ff ff 99 d6 99 d6 99 d6 99 d6 00 00 00 00 99 d6 ..............
000005d0 99 d6 00 00 00 00 99 d6 99 d6 99 d6 10 84 08 42 ..............B
000005e0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 00 00 .B..............
000005f0 00 00 00 00 00 00 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000600 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 00 00 00 00 ..............
00000610 00 00 00 00 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
00000620 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000630 00 00 00 00 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000640 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 00 00 ..............
00000650 00 00 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
00000660 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 00 00 .B..............
00000670 00 00 00 00 00 00 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000680 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 00 00 00 00 ..............
00000690 00 00 00 00 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
000006a0 08 42 10 84 99 d6 99 d6 99 d6 99 d6 00 00 00 00 .B..............
000006b0 99 d6 99 d6 00 00 00 00 99 d6 99 d6 99 d6 ff ff ..............
000006c0 ff ff 99 d6 99 d6 99 d6 99 d6 00 00 00 00 99 d6 ..............
000006d0 99 d6 00 00 00 00 99 d6 99 d6 99 d6 10 84 08 42 ..............B
000006e0 08 42 10 84 99 d6 99 d6 99 d6 00 00 00 00 99 d6 .B..............
000006f0 99 d6 99 d6 99 d6 00 00 00 00 99 d6 99 d6 ff ff ..............
00000700 ff ff 99 d6 99 d6 99 d6 00 00 00 00 99 d6 99 d6 ..............
00000710 99 d6 99 d6 00 00 00 00 99 d6 99 d6 10 84 08 42 ..............B
00000720 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000730 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
00000740 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ..............
00000750 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ..............B
00000760 08 42 10 84 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 .B..............
00000770 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ff ff ..............
```

```
00000780 ff ff 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 ................
00000790 99 d6 99 d6 99 d6 99 d6 99 d6 99 d6 10 84 08 42 ...............B
000007a0 08 42 10 84 10 84 10 84 10 84 10 84 10 84 10 84 .B..............
000007b0 10 84 10 84 10 84 10 84 10 84 10 84 99 d6 ff ff ................
000007c0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ................
000007d0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff 08 42 ...............B
000007e0 08 42 08 42 08 42 08 42 08 42 08 42 08 42 08 42 .B.B.B.B.B.B.B.B
000007f0 08 42 08 42 08 42 08 42 08 42 08 42 08 42 ff ff .B.B.B.B.B.B.B..
```

4.2.2 Cache Color Table

The following is an annotated dump of a Cache Color Table (section 2.2.2.3.1.2.4) Secondary Drawing Order.

```
00000000 03 fc 03 00 00 01 00 00 01 00 00 00 00 00 00 80 ................
00000010 00 00 80 00 00 00 80 80 00 80 00 00 00 80 00 80 ................
00000020 00 80 80 00 00 c0 c0 c0 00 c0 dc c0 00 f0 ca a6 ................
00000030 00 01 1f 3f 00 01 1f 5f 00 01 1f 7f 00 01 1f 9f ...?..._........
00000040 00 01 1f bf 00 01 1f df 00 01 3f 01 00 01 3f 1f ..........?...?.
00000050 00 01 3f 3f 00 01 3f 5f 00 01 3f 7f 00 01 3f 9f ..??..? ..?...?.
00000060 00 01 3f bf 00 01 3f df 00 01 5f 01 00 01 5f 1f ..?...?... ... .
00000070 00 01 5f 3f 00 01 5f 5f 00 01 5f 7f 00 01 5f 9f .. ?..  .. ... .
00000080 00 01 5f bf 00 01 5f df 00 01 7f 01 00 01 7f 1f ..._..._........
00000090 00 01 7f 3f 00 01 7f 5f 00 01 7f 7f 00 01 7f 9f ...?..._........
000000a0 00 01 7f bf 00 01 7f df 00 01 9f 01 00 01 9f 1f ................
000000b0 00 01 9f 3f 00 01 9f 5f 00 01 9f 7f 00 01 9f 9f ...?... ........
000000c0 00 01 9f bf 00 01 9f df 00 01 bf 01 00 01 bf 1f ................
000000d0 00 01 bf 3f 00 01 bf 5f 00 01 bf 7f 00 01 bf 9f ...?... ........
000000e0 00 01 bf bf 00 01 bf df 00 01 df 01 00 01 df 1f ................
000000f0 00 01 df 3f 00 01 df 5f 00 01 df 7f 00 01 df 9f ...?..._........
00000100 00 01 df bf 00 01 df df 00 3f 01 01 00 3f 01 1f .........?...?..
00000110 00 3f 01 3f 00 3f 01 5f 00 3f 01 7f 00 3f 01 9f .?.?.?. .?...?..
00000120 00 3f 01 bf 00 3f 01 df 00 3f 1f 01 00 3f 1f 1f .?...?...?...?..
00000130 00 3f 1f 3f 00 3f 1f 5f 00 3f 1f 7f 00 3f 1f 9f .?.?.?. .?...?..
00000140 00 3f 1f bf 00 3f 1f df 00 3f 3f 01 00 3f 3f 1f .?...?...??..??.
00000150 00 3f 3f 3f 00 3f 3f 5f 00 3f 3f 7f 00 3f 3f 9f .???.??_.??..??.
00000160 00 3f 3f bf 00 3f 3f df 00 3f 5f 01 00 3f 5f 1f .??..??..? ..? .
00000170 00 3f 5f 3f 00 3f 5f 5f 00 3f 5f 7f 00 3f 5f 9f .? ?.?  .? ..? .
00000180 00 3f 5f bf 00 3f 5f df 00 3f 7f 01 00 3f 7f 1f .? ..? ..?...?..
00000190 00 3f 7f 3f 00 3f 7f 5f 00 3f 7f 7f 00 3f 7f 9f .?.?.?. .?...?..
000001a0 00 3f 7f bf 00 3f 7f df 00 3f 9f 01 00 3f 9f 1f .?...?...?...?..
000001b0 00 3f 9f 3f 00 3f 9f 5f 00 3f 9f 7f 00 3f 9f 9f .?.?.?._.?...?..
000001c0 00 3f 9f bf 00 3f 9f df 00 3f bf 01 00 3f bf 1f .?...?...?...?..
000001d0 00 3f bf 3f 00 3f bf 5f 00 3f bf 7f 00 3f bf 9f .?.?.?. .?...?..
000001e0 00 3f bf bf 00 3f bf df 00 3f df 01 00 3f df 1f .?...?...?...?..
000001f0 00 3f df 3f 00 3f df 5f 00 3f df 7f 00 3f df 9f .?.?.?. .?...?..
00000200 00 3f df bf 00 3f df df 00 7f 01 01 00 7f 01 1f .?...?.........
00000210 00 7f 01 3f 00 7f 01 5f 00 7f 01 7f 00 7f 01 9f ...?..._........
00000220 00 7f 01 bf 00 7f 01 df 00 7f 1f 01 00 7f 1f 1f ................
00000230 00 7f 1f 3f 00 7f 1f 5f 00 7f 1f 7f 00 7f 1f 9f ...?... ........
00000240 00 7f 1f bf 00 7f 1f df 00 7f 3f 01 00 7f 3f 1f ..........?...?.
00000250 00 7f 3f 3f 00 7f 3f 5f 00 7f 3f 7f 00 7f 3f 9f ..??..? ..?...?.
00000260 00 7f 3f bf 00 7f 3f df 00 7f 5f 01 00 7f 5f 1f ..?...?..._..._.
00000270 00 7f 5f 3f 00 7f 5f 5f 00 7f 5f 7f 00 7f 5f 9f .._?.._.._..._.
00000280 00 7f 5f bf 00 7f 5f df 00 7f 7f 01 00 7f 7f 1f .. ... .........
00000290 00 7f 7f 3f 00 7f 7f 5f 00 7f 7f 7f 00 7f 7f 9f ...?... ........
000002a0 00 7f 7f bf 00 7f 7f df 00 7f 9f 01 00 7f 9f 1f ................
000002b0 00 7f 9f 3f 00 7f 9f 5f 00 7f 9f 7f 00 7f 9f 9f ...?..._........
000002c0 00 7f 9f bf 00 7f 9f df 00 7f bf 01 00 7f bf 1f ................
```

```
000002d0 00 7f bf 3f 00 7f bf 5f 00 7f bf 7f 00 7f bf 9f ...?... ........
000002e0 00 7f bf bf 00 7f bf df 00 7f df 01 00 7f df 1f ................
000002f0 00 7f df 3f 00 7f df 5f 00 7f df 7f 00 7f df 9f ...?..._........
00000300 00 7f df bf 00 7f df df 00 bf 01 01 00 bf 01 1f ................
00000310 00 bf 01 3f 00 bf 01 5f 00 bf 01 7f 00 bf 01 9f ...?... ........
00000320 00 bf 01 bf 00 bf 01 df 00 bf 1f 01 00 bf 1f 1f ................
00000330 00 bf 1f 3f 00 bf 1f 5f 00 bf 1f 7f 00 bf 1f 9f ...?... ........
00000340 00 bf 1f bf 00 bf 1f df 00 bf 3f 01 00 bf 3f 1f ..........?...?.
00000350 00 bf 3f 3f 00 bf 3f 5f 00 bf 3f 7f 00 bf 3f 9f ..??..?_..?...?.
00000360 00 bf 3f bf 00 bf 3f df 00 bf 5f 01 00 bf 5f 1f ..?...?... ... .
00000370 00 bf 5f 3f 00 bf 5f 5f 00 bf 5f 7f 00 bf 5f 9f .. ?..  .. ... .
00000380 00 bf 5f bf 00 bf 5f df 00 bf 7f 01 00 bf 7f 1f .. ... .........
00000390 00 bf 7f 3f 00 bf 7f 5f 00 bf 7f 7f 00 bf 7f 9f ...?... ........
000003a0 00 bf 7f bf 00 bf 7f df 00 bf 9f 01 00 bf 9f 1f ................
000003b0 00 bf 9f 3f 00 bf 9f 5f 00 bf 9f 7f 00 bf 9f 9f ...?..._........
000003c0 00 bf 9f bf 00 bf 9f df 00 bf bf 01 00 bf bf 1f ................
000003d0 00 bf bf 3f 00 bf bf 5f 00 bf bf 7f 00 bf bf 9f ...?... ........
000003e0 00 f0 fb ff 00 a4 a0 a0 00 80 80 80 00 00 00 ff ................
000003f0 00 00 ff 00 00 00 ff ff 00 ff 00 00 00 ff 00 ff ................
00000400 00 ff ff 00 00 ff ff ff 00                      .........
```

```
03 -> SECONDARY DRAWING ORDER HEADER::controlFlags = 0x03
0x03 =
  0x01 |
  0x02
= TS_STANDARD |
  TS_SECONDARY

fc 03 -> SECONDARY DRAWING ORDER HEADER::orderLength = 0x3fc + 13 = 1033 bytes
00 00 -> SECONDARY DRAWING ORDER HEADER::extraFlags = 0
01 -> CACHE COLOR TABLE ORDER::orderType = TS CACHE COLOR TABLE

00 -> CACHE_COLOR_TABLE_ORDER::cacheIndex
00 01 -> CACHE COLOR TABLE ORDER::numberColors = 0x0100 = 256 Colors

CACHE COLOR TABLE ORDER::ColorTable:
R = red
G = green
B = blue
P = padding

Index  R  G  B  P
0000  00 00 00 00
0004  00 00 80 00
0008  00 80 00 00
000c  00 80 80 00
0010  80 00 00 00
0014  80 00 80 00
0018  80 80 00 00
001c  c0 c0 c0 00
0020  c0 dc c0 00
0024  f0 ca a6 00
0028  01 1f 3f 00
002c  01 1f 5f 00
0030  01 1f 7f 00
0034  01 1f 9f 00
0038  01 1f bf 00
003c  01 1f df 00
0040  01 3f 01 00
0044  01 3f 1f 00
0048  01 3f 3f 00
004c  01 3f 5f 00
0050  01 3f 7f 00
0054  01 3f 9f 00
0058  01 3f bf 00
```

```
005c  01 3f df 00
0060  01 5f 01 00
0064  01 5f 1f 00
0068  01 5f 3f 00
006c  01 5f 5f 00
0070  01 5f 7f 00
0074  01 5f 9f 00
0078  01 5f bf 00
007c  01 5f df 00
0080  01 7f 01 00
0084  01 7f 1f 00
0088  01 7f 3f 00
008c  01 7f 5f 00
0090  01 7f 7f 00
0094  01 7f 9f 00
0098  01 7f bf 00
009c  01 7f df 00
00a0  01 9f 01 00
00a4  01 9f 1f 00
00a8  01 9f 3f 00
00ac  01 9f 5f 00
00b0  01 9f 7f 00
00b4  01 9f 9f 00
00b8  01 9f bf 00
00bc  01 9f df 00
00c0  01 bf 01 00
00c4  01 bf 1f 00
00c8  01 bf 3f 00
00cc  01 bf 5f 00
00d0  01 bf 7f 00
00d4  01 bf 9f 00
00d8  01 bf bf 00
00dc  01 bf df 00
00e0  01 df 01 00
00e4  01 df 1f 00
00e8  01 df 3f 00
00ec  01 df 5f 00
00f0  01 df 7f 00
00f4  01 df 9f 00
00f8  01 df bf 00
00fc  01 df df 00
0100  3f 01 01 00
0104  3f 01 1f 00
0108  3f 01 3f 00
010c  3f 01 5f 00
0110  3f 01 7f 00
0114  3f 01 9f 00
0118  3f 01 bf 00
011c  3f 01 df 00
0120  3f 1f 01 00
0124  3f 1f 1f 00
0128  3f 1f 3f 00
012c  3f 1f 5f 00
0130  3f 1f 7f 00
0134  3f 1f 9f 00
0138  3f 1f bf 00
013c  3f 1f df 00
0140  3f 3f 01 00
0144  3f 3f 1f 00
0148  3f 3f 3f 00
014c  3f 3f 5f 00
0150  3f 3f 7f 00
0154  3f 3f 9f 00
0158  3f 3f bf 00
015c  3f 3f df 00
```

```
0160  3f 5f 01 00
0164  3f 5f 1f 00
0168  3f 5f 3f 00
016c  3f 5f 5f 00
0170  3f 5f 7f 00
0174  3f 5f 9f 00
0178  3f 5f bf 00
017c  3f 5f df 00
0180  3f 7f 01 00
0184  3f 7f 1f 00
0188  3f 7f 3f 00
018c  3f 7f 5f 00
0190  3f 7f 7f 00
0194  3f 7f 9f 00
0198  3f 7f bf 00
019c  3f 7f df 00
01a0  3f 9f 01 00
01a4  3f 9f 1f 00
01a8  3f 9f 3f 00
01ac  3f 9f 5f 00
01b0  3f 9f 7f 00
01b4  3f 9f 9f 00
01b8  3f 9f bf 00
01bc  3f 9f df 00
01c0  3f bf 01 00
01c4  3f bf 1f 00
01c8  3f bf 3f 00
01cc  3f bf 5f 00
01d0  3f bf 7f 00
01d4  3f bf 9f 00
01d8  3f bf bf 00
01dc  3f bf df 00
01e0  3f df 01 00
01e4  3f df 1f 00
01e8  3f df 3f 00
01ec  3f df 5f 00
01f0  3f df 7f 00
01f4  3f df 9f 00
01f8  3f df bf 00
01fc  3f df df 00
0200  7f 01 01 00
0204  7f 01 1f 00
0208  7f 01 3f 00
020c  7f 01 5f 00
0210  7f 01 7f 00
0214  7f 01 9f 00
0218  7f 01 bf 00
021c  7f 01 df 00
0220  7f 1f 01 00
0224  7f 1f 1f 00
0228  7f 1f 3f 00
022c  7f 1f 5f 00
0230  7f 1f 7f 00
0234  7f 1f 9f 00
0238  7f 1f bf 00
023c  7f 1f df 00
0240  7f 3f 01 00
0244  7f 3f 1f 00
0248  7f 3f 3f 00
024c  7f 3f 5f 00
0250  7f 3f 7f 00
0254  7f 3f 9f 00
0258  7f 3f bf 00
025c  7f 3f df 00
0260  7f 5f 01 00
```

```
0264  7f 5f 1f 00
0268  7f 5f 3f 00
026c  7f 5f 5f 00
0270  7f 5f 7f 00
0274  7f 5f 9f 00
0278  7f 5f bf 00
027c  7f 5f df 00
0280  7f 7f 01 00
0284  7f 7f 1f 00
0288  7f 7f 3f 00
028c  7f 7f 5f 00
0290  7f 7f 7f 00
0294  7f 7f 9f 00
0298  7f 7f bf 00
029c  7f 7f df 00
02a0  7f 9f 01 00
02a4  7f 9f 1f 00
02a8  7f 9f 3f 00
02ac  7f 9f 5f 00
02b0  7f 9f 7f 00
02b4  7f 9f 9f 00
02b8  7f 9f bf 00
02bc  7f 9f df 00
02c0  7f bf 01 00
02c4  7f bf 1f 00
02c8  7f bf 3f 00
02cc  7f bf 5f 00
02d0  7f bf 7f 00
02d4  7f bf 9f 00
02d8  7f bf bf 00
02dc  7f bf df 00
02e0  7f df 01 00
02e4  7f df 1f 00
02e8  7f df 3f 00
02ec  7f df 5f 00
02f0  7f df 7f 00
02f4  7f df 9f 00
02f8  7f df bf 00
02fc  7f df df 00
0300  bf 01 01 00
0304  bf 01 1f 00
0308  bf 01 3f 00
030c  bf 01 5f 00
0310  bf 01 7f 00
0314  bf 01 9f 00
0318  bf 01 bf 00
031c  bf 01 df 00
0320  bf 1f 01 00
0324  bf 1f 1f 00
0328  bf 1f 3f 00
032c  bf 1f 5f 00
0330  bf 1f 7f 00
0334  bf 1f 9f 00
0338  bf 1f bf 00
033c  bf 1f df 00
0340  bf 3f 01 00
0344  bf 3f 1f 00
0348  bf 3f 3f 00
034c  bf 3f 5f 00
0350  bf 3f 7f 00
0354  bf 3f 9f 00
0358  bf 3f bf 00
035c  bf 3f df 00
0360  bf 5f 01 00
0364  bf 5f 1f 00
```

```
0368  bf 5f 3f 00
036c  bf 5f 5f 00
0370  bf 5f 7f 00
0374  bf 5f 9f 00
0378  bf 5f bf 00
037c  bf 5f df 00
0380  bf 7f 01 00
0384  bf 7f 1f 00
0388  bf 7f 3f 00
038c  bf 7f 5f 00
0390  bf 7f 7f 00
0394  bf 7f 9f 00
0398  bf 7f bf 00
039c  bf 7f df 00
03a0  bf 9f 01 00
03a4  bf 9f 1f 00
03a8  bf 9f 3f 00
03ac  bf 9f 5f 00
03b0  bf 9f 7f 00
03b4  bf 9f 9f 00
03b8  bf 9f bf 00
03bc  bf 9f df 00
03c0  bf bf 01 00
03c4  bf bf 1f 00
03c8  bf bf 3f 00
03cc  bf bf 5f 00
03d0  bf bf 7f 00
03d4  bf bf 9f 00
03d8  f0 fb ff 00
03dc  a4 a0 a0 00
03e0  80 80 80 00
03e4  00 00 ff 00
03e8  00 ff 00 00
03ec  00 ff ff 00
03f0  ff 00 00 00
03f4  ff 00 ff 00
03f8  ff ff 00 00
03fc  ff ff ff 00
```

4.2.3 Cache Glyph (Revision 2)

The following is an annotated dump of a Cache Glyph (Revision 2) (section 2.2.2.3.1.2.6) Secondary Drawing Order.

```
00000000 03 04 00 37 01 03 34 00 49 03 03 a0 a0 a0 00 22  ...7..4.I......"
00000010 00                                               .

03 -> SECONDARY_DRAWING_ORDER_HEADER::controlFlags = 0x03
0x03
= 0x2 |
  0x1
= TS SECONDARY |
  TS_STANDARD

04 00 -> SECONDARY DRAWING ORDER HEADER::orderLength = 0x04 + 13 = 17 bytes

37 01 -> SECONDARY DRAWING ORDER HEADER::extraFlags = 0x137 = binary:00000001 0011 0111

cacheId = binary:0111 = 7
```

```
flags = binary:0011 = 0x03
0x03
= 0x01 |
  0x02
= CG2 GLYPH UNICODE PRESENT |
  CG2 GLYPH REV2

cGlyphs = 0000 0001 = 0x1 = 1

03 -> SECONDARY DRAWING ORDER HEADER::header::orderType = 0x03 = TS CACHE GLYPH

34 -> TS CACHE GLYPH DATA REV2::cacheIndex = 0x34 = 52
00 -> TS_CACHE_GLYPH_DATA_REV2::x = 0
49 -> TS_CACHE_GLYPH_DATA_REV2::y = -9
03 -> TS CACHE GLYPH DATA REV2::cx = 3
03 -> TS CACHE GLYPH DATA REV2::cy = 3

a0 a0 a0 00 -> TS CACHE GLYPH DATA REV2::aj

a0 -> X.X
a0 -> X.X
a0 -> X.X

00 00 -> padding

22 00 -> CACHE_GLYPH_REV2_ORDER::unicodeCharacters = 0x22 = "
```

4.2.4 Cache Brush

The following is an annotated dump of a Cache Brush (section 2.2.2.3.1.2.7) Secondary Drawing Order.

```
00000000 03 07 00 00 00 07 00 01 08 08 81 08 aa 55 aa 55  .............U.U
00000010 aa 55 aa 55                                      .U.U

03 -> SECONDARY DRAWING ORDER HEADER::controlFlags = 0x03
0x03
= 0x2 |
  0x1
= TS_SECONDARY |
  TS_STANDARD

07 00 -> SECONDARY DRAWING ORDER HEADER::orderLength = 0x7 + 13 = 20 bytes
00 00 -> SECONDARY DRAWING ORDER HEADER::extraFlags = 0x0
07 -> SECONDARY DRAWING ORDER HEADER::orderType = 0x07 = TS CACHE BRUSH

00 -> CACHE_BRUSH_ORDER::cacheEntry = 0
01 -> CACHE BRUSH ORDER::iBitmapFormat = TS BMF 1BPP (0x01)
08 -> CACHE BRUSH ORDER::cx = 0x8 = 8
08 -> CACHE BRUSH ORDER::cy = 0x8 = 8
81 -> CACHE BRUSH ORDER::Style = 0x81

08 -> CACHE_BRUSH_ORDER::iBytes = 0x8 = 8 bytes

aa 55 aa 55 aa 55 aa 55 -> CACHE BRUSH ORDER::brushData

0xaa -> 1 0 1 0 1 0 1 0
0x55 -> 0 1 0 1 0 1 0 1
```

```
0xaa -> 1 0 1 0 1 0 1 0
0x55 -> 0 1 0 1 0 1 0 1
0xaa -> 1 0 1 0 1 0 1 0
0x55 -> 0 1 0 1 0 1 0 1
0xaa -> 1 0 1 0 1 0 1 0
0x55 -> 0 1 0 1 0 1 0 1
```

4.3 Annotated Alternate Secondary Drawing Orders

4.3.1 Create Offscreen Bitmap

The following is an annotated dump of a Create Offscreen Bitmap (section 2.2.2.3.1.3.2) Alternate Secondary Drawing Order.

```
00000000 06 00 80 60 01 10 00 01 00 02 00    ...`.......

06 -> ALTSEC_DRAWING_ORDER_HEADER::controlFlags = 0x06 = binary:000001 10
class = TS_SECONDARY (0x02)
orderType =  TS ALTSEC CREATE OFFSCR BITMAP (0x01)

00 80 -> CREATE OFFSCR BITMAP ORDER::flags = 0x8000
OffscreenBitmapId = 0
d = 1

60 01 -> CREATE OFFSCR BITMAP ORDER::cx = 0x0160 = 352
10 00 -> CREATE OFFSCR BITMAP ORDER::cy = 0x0010 = 16

01 00 -> OFFSCR_DELETE_LIST::cIndices = 1
02 00 -> OFFSCR_DELETE_LIST::indices[0] = 0x0002
```

4.3.2 Switch Surface

The following is an annotated dump of a Switch Surface (section 2.2.2.3.1.3.3) Alternate Secondary Drawing Order.

```
00000000 02 ff ff                                    ...

02 -> ALTSEC DRAWING ORDER HEADER::controlFlags = 0x02 = binary:000000 10
class = TS SECONDARY (0x02)
orderType = TS ALTSEC SWITCH SURFACE (0x00)

ff ff -> SWITCH_SURFACE_ORDER::bitmapId = SCREEN_BITMAP_SURFACE (0xFFFF)
```

4.3.3 Create NineGrid Bitmap

The following is an annotated dump of a Create NineGrid Bitmap (section 2.2.2.3.1.3.4) Alternate Secondary Drawing Order.

```
00000000 12 20 41 00 08 00 ab 01 05 00 00 00 00 00 07 00  . A.............
00000010 07 00 02 00 01 00 00 00                          ........

12 -> ALTSEC_DRAWING_ORDER_HEADER::controlFlags = 0x12
```

```
class = TS SECONDARY (0x02)
orderType = TS_ALTSEC_CREATE_NINEGRID_BITMAP  (0x04)

20 -> CREATE NINEGRID BITMAP ORDER::BitmapBpp = 0x20 = 32 Bpp
41 00 -> CREATE NINEGRID BITMAP ORDER::BitmapId = 0x41 = 65
08 00 -> CREATE NINEGRID BITMAP ORDER::cx = 0x8 = 8
ab 01 -> CREATE NINEGRID BITMAP ORDER::cy = 0x01ab = 427

05 00 00 00 -> NINEGRID_BITMAP_INFO::flFlags = 0x5
0x5
= 0x1 |
  0x4
= DSDNG STRETCH |
  DSDNG_PERPIXELALPHA

00 00 -> NINEGRID BITMAP INFO::ulLeftWidth = 0x0 = 0
07 00 -> NINEGRID BITMAP INFO::ulRightWidth = 0x7 = 7
07 00 -> NINEGRID BITMAP INFO::ulTopHeight = 0x7 = 7
02 00 -> NINEGRID BITMAP INFO::ulBottomHeight = 0x2 = 2

01 00 00 00 -> NINEGRID BITMAP INFO::crTransparent = 0x00000001
01 -> TS COLORREF::red = 0x1 = 1
00 -> TS COLORREF::green = 0
00 -> TS COLORREF::blue = 0
00 -> TS_COLORREF::zeroPad = 0
```

4.3.4 Stream Bitmap First

The following is an annotated dump of a Stream Bitmap First (section 2.2.2.3.1.3.5.1) Alternate Secondary Drawing Order.

```
00000000 0a 07 20 01 00 04 00 1e 00 7a 01 00 00 7a 01 02 .. ......z...z..
00000010 1c df 60 02 90 df 60 02 fc 19 f5 6c 8f 6a 9a 02 ..`...`....l.j..
00000020 5f 07 00 00 88 df 60 02 02 00 00 00 a8 e6 8d 02 _.....`.........
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000050 00 00 00 00 01 00 00 00 01 00 00 00 84 df 60 02 ..............`.
00000060 6f e3 28 e7 08 00 00 00 05 40 00 80 00 00 00 00 o.(......@......
00000070 ee 71 9a 02 a8 e0 60 02 e3 57 ec 6c 02 00 00 00 .q....`..W.l....
00000080 8f 6a 9a 02 5f 07 00 00 90 e0 60 02 04 00 00 00 .j.._.....`.....
00000090 05 73 9a 02 1f e7 99 02 1e 00 00 00 01 00 00 00 .s..............
000000a0 a8 e6 8d 02 01 00 00 00 d0 df 60 02 02 00 00 00 ..........`.....
000000b0 fc c3 38 00 ff 17 00 00 10 2f 99 02 0f 73 9a 02 ..8....../...s..
000000c0 00 00 00 00 f0 2e 99 02 ba ed 7c 02 00 00 00 01 ..........|.....
000000d0 8d 6a 9a 02 ff ff 00 00 00 00 00 00 00 2f 99 02 .j.........../..
000000e0 48 52 f8 00 0d 00 00 00 00 00 00 00 04 00 00 00 HR..............
000000f0 ba ed 7c 02 00 00 00 00 0a 00 00 00 00 01 00 00 ..|.............
00000100 0a 00 00 00 10 33 99 02 ee 71 9a 02 10 73 99 02 .....3...q...s..
00000110 ff 1f 00 00 ff ff 00 00 ff ff 00 00 68 e0 60 02 ............h.`.
00000120 1a 4c ed 6c d0 ec 7c 02 ea 00 00 00 00 00 00 00 .L.l..|.........
00000130 b8 e1 60 02 a8 e1 60 02 f0 2e 99 02 02 00 00 00 ..`...`.........
00000140 02 00 00 00 04 00 00 00 87 dc 28 e7 b0 e0 60 02 ..........(...`.
00000150 01 62 e5 6c 08 00 00 00 04 00 00 00 00 00 00 00 .b.l............
00000160 08 00 00 00 8f 6a 9a 02 8f 6a 9a 02 5f 07 00 00 .....j...j.. ...
00000170 00 00 00 00 c8 d8 7a 02 ee 71 9a 02 00 00 00 00 ......z..q......
00000180 00 00 00 00 4b dc 28 e7 02                      ....K.(..
```

```
0a -> ALTSEC_DRAWING_ORDER_HEADER::controlFlags = 0xa
class = TS SECONDARY (0x02)
orderType = TS ALTSEC STREAM BITMAP FIRST (0x02)

07 -> STREAM_BITMAP_FIRST_ORDER::BitmapFlags = 0x07
0x07
```

```
= 0x01 |
  0x02 |
  0x04
= TS STREAM BITMAP END |
  TS STREAM BITMAP COMPRESSED |
  TS STREAM BITMAP REV2

20 -> STREAM_BITMAP_FIRST_ORDER::BitmapBpp = 0x20 = 32 bpp
01 00 -> STREAM_BITMAP_FIRST_ORDER::BitmapType = TS_DRAW_NINEGRID_BITMAP_CACHE (0x1)
04 00 -> STREAM BITMAP FIRST ORDER::BitmapWidth = 0x4 = 4
1e 00 -> STREAM BITMAP FIRST ORDER::BitmapHeight = 0x1e = 30
7a 01 00 00 -> STREAM BITMAP FIRST ORDER::BitmapSize = 0x17a = 378 bytes
7a 01 -> STREAM BITMAP FIRST ORDER::BitmapBlockSize = 0x17a = 378 bytes

The next 378 bytes are the compressed bitmap (STREAM_BITMAP_FIRST_ORDER::BitmapBlock).
```

4.3.5 Stream Bitmap Next

The following is an annotated dump of a Stream Bitmap Next (section 2.2.2.3.1.3.5.2) Alternate Secondary Drawing Order.

```
00000000 0e 02 01 00 00 10 ff 03 71 f8 f8 f8 ff 03 c0 01  ........q.......
00000010 12 12 12 00 03 c0 01 1b 1b 1b 00 03 c0 01 1e 1e  ................
00000020 1e 00 03 85 ed ed ed ff ed ed ed ff ed ed ed ff  ................
00000030 ed ed ed ff ed ed ed ff cc 01 01 01 00 00 97 6c  ...............l
00000040 eb eb eb ff 8d ec                                ......
...
00000fc0 ff fe fd fb ff fe fd fb ff fe fd fb ff fe fd fb  ................
00000fd0 ff fe fd fb ff fe fc fa ff eb e2 d5 ff b8 8f 59  ...............Y
00000fe0 ff ea ea ea ff 00 00 00 00 00 00 00 00 00 00 00  ................
00000ff0 00 f8 f8 f8 ff d3 c3 ae ff bf 9a 69 ff 6b b1 7f  ...........i.k..
00001000 3c ff 95 bc 93 5d                               <....]

0e -> ALTSEC_DRAWING_ORDER_HEADER::controlFlags = 0x0e
class = TS SECONDARY (0x02)
orderType = TS ALTSEC STREAM BITMAP NEXT (0x03)

02 -> STREAM BITMAP NEXT ORDER::BitmapFlags = 0x2 = TS STREAM BITMAP COMPRESSED
01 00 -> STREAM_BITMAP_NEXT_ORDER::BitmapType = TS_DRAW_NINEGRID_BITMAP_CACHE (0x1)
00 10 -> STREAM_BITMAP_NEXT_ORDER::BitmapBlockSize = 0x1000 = 4096 bytes

The next 4096 bytes are the compressed bitmap (STREAM_BITMAP_FIRST_ORDER::BitmapBlock).
```

4.4 Standard Security Server Redirection PDU

The following is an annotated dump of a Standard Security Server Redirection PDU, as specified in section 2.2.3.1.

```
00000000 03 00 02 1f 02 f0 80 68 00 01 03 eb 70 82 10 00 .......h....p...
00000010 0c 00 00 58 dd 3f e5 f3 de 80 26 c0 d6 3f 26 0e ...X.?....&..?&.
00000020 2c b5 93 dd 26 d5 4b 84 a1 1d 2a 78 85 38 cf 1d ,...&.K...*x.8..
00000030 72 80 46 0e 72 fb fd 29 77 e7 e3 0a ba 3f cc a4 r.F.r..)w....?..
00000040 50 2c 5b 87 cb e2 2b 61 ea 9a b7 19 25 a6 ea 33 P,[...+a....%..3
00000050 01 9a 2e 3a 58 fe 7e 1e 66 c0 3c a0 d3 5b d1 96 ...:X.~.f.<..[..
00000060 43 4a f4 94 57 b2 71 ba df 69 ed 3a ad b2 83 a5 CJ..W.q..i.:....
00000070 d8 db 8d e1 c1 5e 73 6c d3 61 3c fc ae 05 78 94 .....^sl.a<...x.
00000080 f2 f6 87 ae 78 24 8e 5b 50 d6 36 2c c6 56 e2 2d ....x$.[P.6,.V.-
00000090 61 46 d3 a3 22 d6 ce 1a 26 1c 1e e0 9b 97 2d 98 aF.."...&.....-.
000000a0 45 3c b9 92 47 1a 25 f0 8c 7c c0 6f 54 b6 09 21 E<..G.%..|.oT..!
000000b0 67 e3 41 3e 4e b9 be d2 86 d9 38 10 69 d7 f5 90 g.A>N.....8.i...
```

```
000000c0 ef c1 50 39 13 b2 9b 7c 98 52 35 0f 90 26 cc ad ..P9...|.R5..&..
000000d0 7d df 11 37 97 09 d9 69 12 0a 5f 3b bd 38 28 f6 }..7...i.._;.8(.
000000e0 8a 4d 65 a6 3f 74 8f 6d 09 84 e2 03 b6 35 b9 b1 .Me.?t.m.....5..
000000f0 11 10 b0 53 5e c8 25 f0 b2 bd af 4c ce 49 62 de ...S^.%....L.Ib.
00000100 23 67 43 66 0a f1 3a 8f d7 9d 80 fb 2a 37 c3 de #gCf..:.....*7..
00000110 8e 02 16 e2 12 73 2b 58 b8 5e 7e 61 ba 6f 80 73 .....s+X.^~a.o.s
00000120 0b f5 27 b7 45 1c bf 6a 1c fe 74 55 df 81 f6 06 ..'.E..j..tU....
00000130 f3 ca b2 ce a8 d4 94 75 24 c2 02 0a 56 a9 fd 13 .......u$...V...
00000140 a6 af 8d 53 66 49 4d 4e bc b2 ff 80 5b 48 68 da ...SfIMN....[Hh.
00000150 ee 01 1c bd a2 17 42 50 e5 15 4e 21 0c 6e d3 5b ......BP..N!.n.[
00000160 3c 5a ce bc 0f e3 13 fb a3 7f 3c e0 7a c7 be 06 <Z........<.z...
00000170 90 7a a2 91 33 ce 00 68 21 63 89 a3 5c 43 be 96 .z..3..h!c..\C..
00000180 e0 11 b8 48 a8 47 1a 75 47 22 2f 3f 97 8d bd 14 ...H.G.uG"/?....
00000190 34 a5 89 06 49 6a 8c 19 82 eb 4f 7e ec 06 80 e2 4...Ij....O~....
000001a0 20 b5 ac 04 65 da 98 65 27 8f 45 80 ff 73 3e af  ...e..e'.E..s>.
000001b0 05 ab bc e4 66 4d d0 34 85 a5 9a a4 57 5a c6 b9 ....fM.4....WZ..
000001c0 27 e7 73 37 7e 7c 0b 65 24 cd 5c 61 89 f7 13 a2 '.s7~|.e$.\a....
000001d0 d8 e1 85 ea 6f 81 7a 3b f5 e8 fb 45 92 f2 81 8c ....o.z;...E....
000001e0 cd 59 84 13 d9 6b db 0a ba af 0c 4f 9a de aa d6 .Y...k.....O....
000001f0 a1 44 db cc 07 4c 71 4e 2a c3 50 9c f5 0f 9e 2b .D...LqN*.P....+
00000200 2f 4b bb b6 fa 08 d1 65 e3 1a 1a 62 06 c4 ec 41 /K.....e...b...A
00000210 69 6b d5 86 93 9c 46 de 4f 07 11 55 54 e9 16    ik....F.O..UT..

03 00 02 1f -> TPKT Header (length = 543 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 82 10 -> PER encoded (basic aligned variant) SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x210 = 528 bytes

00 0c -> TS SECURITY HEADER::flags = 0x0c00
0x0c00
= 0x0800 | 0x0400
= SEC SECURE CHECKSUM | SEC REDIRECTION PKT

00 00 -> TS_SECURITY_HEADER::flagsHi - ignored as flags field does not contain
RDP SEC FLAGSHI VALID (0x8000)
58 dd 3f e5 f3 de 80 26 -> TS SECURITY HEADER::dataSignature

c0 d6 3f 26 0e 2c b5 93 dd 26 d5 4b 84 a1 1d 2a
78 85 38 cf 1d 72 80 46 0e 72 fb fd 29 77 e7 e3
0a ba 3f cc a4 50 2c 5b 87 cb e2 2b 61 ea 9a b7
19 25 a6 ea 33 01 9a 2e 3a 58 fe 7e 1e 66 c0 3c
a0 d3 5b d1 96 43 4a f4 94 57 b2 71 ba df 69 ed
3a ad b2 83 a5 d8 db 8d e1 c1 5e 73 6c d3 61 3c
fc ae 05 78 94 f2 f6 87 ae 78 24 8e 5b 50 d6 36
2c c6 56 e2 2d 61 46 d3 a3 22 d6 ce 1a 26 1c 1e
e0 9b 97 2d 98 45 3c b9 92 47 1a 25 f0 8c 7c c0
6f 54 b6 09 21 67 e3 41 3e 4e b9 be d2 86 d9 38
10 69 d7 f5 90 ef c1 50 39 13 b2 9b 7c 98 52 35
0f 90 26 cc ad 7d df 11 37 97 09 d9 69 12 0a 5f
3b bd 38 28 f6 8a 4d 65 a6 3f 74 8f 6d 09 84 e2
03 b6 35 b9 b1 11 10 b0 53 5e c8 25 f0 b2 bd af
4c ce 49 62 de 23 67 43 66 0a f1 3a 8f d7 9d 80
fb 2a 37 c3 de 8e 02 16 e2 12 73 2b 58 b8 5e 7e
61 ba 6f 80 73 0b f5 27 b7 45 1c bf 6a 1c fe 74
55 df 81 f6 06 f3 ca b2 ce a8 d4 94 75 24 c2 02
0a 56 a9 fd 13 a6 af 8d 53 66 49 4d 4e bc b2 ff
80 5b 48 68 da ee 01 1c bd a2 17 42 50 e5 15 4e
21 0c 6e d3 5b 3c 5a ce bc 0f e3 13 fb a3 7f 3c
e0 7a c7 be 06 90 7a a2 91 33 ce 00 68 21 63 89
a3 5c 43 be 96 e0 11 b8 48 a8 47 1a 75 47 22 2f
```

```
3f 97 8d bd 14 34 a5 89 06 49 6a 8c 19 82 eb 4f
7e ec 06 80 e2 20 b5 ac 04 65 da 98 65 27 8f 45
80 ff 73 3e af 05 ab bc e4 66 4d d0 34 85 a5 9a
a4 57 5a c6 b9 27 e7 73 37 7e 7c 0b 65 24 cd 5c
61 89 f7 13 a2 d8 e1 85 ea 6f 81 7a 3b f5 e8 fb
45 92 f2 81 8c cd 59 84 13 d9 6b db 0a ba af 0c
4f 9a de aa d6 a1 44 db cc 07 4c 71 4e 2a c3 50
9c f5 0f 9e 2b 2f 4b bb b6 fa 08 d1 65 e3 1a 1a
62 06 c4 ec 41 69 6b d5 86 93 9c 46 de 4f 07 11
55 54 e9 16 -> Encrypted RDP SERVER REDIRECTION PACKET

Decrypted RDP SERVER REDIRECTION PACKET:
00000000 00 04 04 02 02 00 00 00 1d 0b 00 00 46 00 00 00 ............F...
00000010 32 00 30 00 30 00 31 00 3a 00 34 00 38 00 39 00 2.0.0.1.:.4.8.9.
00000020 38 00 3a 00 32 00 62 00 3a 00 32 00 3a 00 39 00 8.:.2.b.:.2.:.9.
00000030 64 00 65 00 37 00 3a 00 34 00 35 00 36 00 39 00 d.e.7.:.4.5.6.9.
00000040 3a 00 66 00 62 00 33 00 39 00 3a 00 65 00 66 00 :.f.b.3.9.:.e.f.
00000050 32 00 39 00 00 00 1c 00 00 00 61 00 64 00 6d 00 2.9.......a.d.m.
00000060 69 00 6e 00 69 00 73 00 74 00 72 00 61 00 74 00 i.n.i.s.t.r.a.t.
00000070 6f 00 72 00 00 00 16 00 00 00 54 00 53 00 2d 00 o.r.......T.S.-.
00000080 53 00 54 00 52 00 45 00 53 00 53 00 31 00 00 00 S.T.R.E.S.S.1...
00000090 78 00 00 00 02 00 00 80 44 53 48 4c 06 6f 27 1b x.......DSHL.o'.
000000a0 29 10 f9 d9 58 fb 46 7d f9 e1 02 14 a2 15 aa 00 )...X.F}........
000000b0 34 5c 76 a4 52 76 fd 04 d6 2d 85 8d 64 69 88 80 4\v.Rv...-..di..
000000c0 1b 8d 0e b0 b7 9b d3 d8 84 c6 10 a2 e9 b6 e0 06 ................
000000d0 99 5d 85 16 2d bf d8 f1 99 77 75 2d be e2 77 a6 .]..-....wu-..w.
000000e0 3f 5e fb 86 ca ed 04 81 31 11 d3 b9 fc 32 ad 45 ?^......1....2.E
000000f0 df ad ca b7 8d 02 6f 92 65 c6 d7 b4 68 cd f6 49 ......o.e...h..I
00000100 bc b8 88 87 6e 01 ce d0 95 fd 00 00 5a 00 00 00 ....n.......Z...
00000110 6a 00 69 00 61 00 7a 00 6f 00 75 00 2d 00 74 00 j.i.a.z.o.u.-.t.
00000120 65 00 73 00 74 00 32 00 2e 00 74 00 73 00 2d 00 e.s.t.2...t.s.-.
00000130 73 00 74 00 72 00 65 00 73 00 73 00 31 00 2e 00 s.t.r.e.s.s.1...
00000140 6e 00 74 00 74 00 65 00 73 00 74 00 2e 00 6d 00 n.t.t.e.s.t...m.
00000150 69 00 63 00 72 00 6f 00 73 00 6f 00 66 00 74 00 i.c.r.o.s.o.f.t.
00000160 2e 00 63 00 6f 00 6d 00 00 00 1a 00 00 00 4a 00 ..c.o.m.......J.
00000170 49 00 41 00 5a 00 4f 00 55 00 2d 00 54 00 45 00 I.A.Z.O.U.-.T.E.
00000180 53 00 54 00 32 00 00 00 70 00 00 00 02 00 00 00 S.T.2...p.......
00000190 46 00 00 00 32 00 30 00 30 00 31 00 3a 00 34 00 F...2.0.0.1.:.4.
000001a0 38 00 39 00 38 00 3a 00 32 00 62 00 3a 00 32 00 8.9.8.:.2.b.:.2.
000001b0 3a 00 39 00 64 00 65 00 37 00 3a 00 34 00 35 00 :.9.d.e.7.:.4.5.
000001c0 36 00 39 00 3a 00 66 00 62 00 33 00 39 00 3a 00 6.9.:.f.b.3.9.:.
000001d0 65 00 66 00 32 00 39 00 00 00 1e 00 00 00 31 00 e.f.2.9.......1.
000001e0 35 00 37 00 2e 00 35 00 39 00 2e 00 32 00 34 00 5.7...5.9...2.4.
000001f0 30 00 2e 00 31 00 34 00 34 00 00 00 c0 c0 c0 c0 0...1.4.4.......
00000200 c0 c0 c0 c0                                     ....

00 04 -> RDP SERVER REDIRECTION PACKET::Flags = 0x0400 = SEC REDIRECTION PKT
04 02 -> RDP SERVER REDIRECTION PACKET::Length = 0x204 = 516 bytes
02 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::SessionID = 2

1d 0b 00 00 -> RDP SERVER REDIRECTION PACKET::RedirFlags = 0x00000b1d
0x00000b1d
= 0x00000800 |
  0x00000200 |
  0x00000100 |
  0x00000010 |
  0x00000008 |
  0x00000004 |
  0x00000001
= LB TARGET NET ADDRESSES |
  LB TARGET NETBIOS NAME |
  LB_TARGET_FQDN |
  LB_PASSWORD |
  LB DOMAIN |
  LB_USERNAME |
```

```
  LB TARGET NET ADDRESS

46 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetAddressLength = 0x46 = 70 bytes

32 00 30 00 30 00 31 00 3a 00 34 00 38 00 39 00
38 00 3a 00 32 00 62 00 3a 00 32 00 3a 00 39 00
64 00 65 00 37 00 3a 00 34 00 35 00 36 00 39 00
3a 00 66 00 62 00 33 00 39 00 3a 00 65 00 66 00
32 00 39 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetAddress =
"2001:4898:2b:2:9de7:4569:fb39:ef29"

1c 00 00 00 -> RDP SERVER REDIRECTION PACKET::UserNameLength = 0x1c = 28

61 00 64 00 6d 00 69 00 6e 00 69 00 73 00 74 00
72 00 61 00 74 00 6f 00 72 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::UserName =
"administrator"

16 00 00 00 -> RDP SERVER REDIRECTION PACKET::DomainLength = 0x16 = 22 bytes

54 00 53 00 2d 00 53 00 54 00 52 00 45 00 53 00
53 00 31 00 00 00 -> RDP SERVER REDIRECTION PACKET::Domain = "TS-STRESS1"

78 00 00 00 -> RDP SERVER REDIRECTION PACKET::PasswordLength = 0x78 = 120 bytes

02 00 00 80 44 53 48 4c 06 6f 27 1b 29 10 f9 d9
58 fb 46 7d f9 e1 02 14 a2 15 aa 00 34 5c 76 a4
52 76 fd 04 d6 2d 85 8d 64 69 88 80 1b 8d 0e b0
b7 9b d3 d8 84 c6 10 a2 e9 b6 e0 06 99 5d 85 16
2d bf d8 f1 99 77 75 2d be e2 77 a6 3f 5e fb 86
ca ed 04 81 31 11 d3 b9 fc 32 ad 45 df ad ca b7
8d 02 6f 92 65 c6 d7 b4 68 cd f6 49 bc b8 88 87
6e 01 ce d0 95 fd 00 00 -> RDP_SERVER_REDIRECTION_PACKET::Password

5a 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetFQDNLength = 0x5a = 90

6a 00 69 00 61 00 7a 00 6f 00 75 00 2d 00 74 00
65 00 73 00 74 00 32 00 2e 00 74 00 73 00 2d 00
73 00 74 00 72 00 65 00 73 00 73 00 31 00 2e 00
6e 00 74 00 74 00 65 00 73 00 74 00 2e 00 6d 00
69 00 63 00 72 00 6f 00 73 00 6f 00 66 00 74 00
2e 00 63 00 6f 00 6d 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetFQDN = "jiazou-
test2.ts-stress1.nttest.microsoft.com"

1a 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetBiosNameLength = 0x1a = 26

4a 00 49 00 41 00 5a 00 4f 00 55 00 2d 00 54 00
45 00 53 00 54 00 32 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetNetBiosName =
"JIAZOU-TEST2"

70 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetAddressesLength = 112 bytes

02 00 00 00 -> TARGET NET ADDRESSES::addressCount = 2

46 00 00 00 -> TARGET NET ADDRESS::addressLength = 70 bytes

32 00 30 00 30 00 31 00 3a 00 34 00 38 00 39 00
38 00 3a 00 32 00 62 00 3a 00 32 00 3a 00 39 00
64 00 65 00 37 00 3a 00 34 00 35 00 36 00 39 00
3a 00 66 00 62 00 33 00 39 00 3a 00 65 00 66 00
32 00 39 00 00 00 -> TARGET NET ADDRESS::address = "2001:4898:2b:2:9de7:4569:fb39:ef29"

1e 00 00 00 -> TARGET_NET_ADDRESS::addressLength = 30 bytes

31 00 35 00 37 00 2e 00 35 00 39 00 2e 00 32 00
```

```
34 00 30 00 2e 00 31 00 34 00 34 00 00 00 -> TARGET NET ADDRESS::address =
"157.59.240.144"

c0 c0 c0 c0 c0 c0 c0 c0 -> Excess padding due to over-allocation bug
```

4.5 Enhanced Security Server Redirection PDU

The following is an annotated dump of an Enhanced Security Server Redirection PDU (section 2.2.3.3.1).

```
00000000 03 00 02 1c 02 f0 80 68 00 01 03 eb 70 82 0d 0d .......h....p...
00000010 02 0a 00 ea 03 5f 59 00 04 04 02 02 00 00 00 1d ..... Y.........
00000020 0b 00 00 46 00 00 00 32 00 30 00 30 00 31 00 3a ...F...2.0.0.1.:
00000030 00 34 00 38 00 39 00 38 00 3a 00 32 00 62 00 3a .4.8.9.8.:.2.b.:
00000040 00 32 00 3a 00 39 00 64 00 65 00 37 00 3a 00 34 .2.:.9.d.e.7.:.4
00000050 00 35 00 36 00 39 00 3a 00 66 00 62 00 33 00 39 .5.6.9.:.f.b.3.9
00000060 00 3a 00 65 00 66 00 32 00 39 00 00 00 1c 00 00 .:.e.f.2.9......
00000070 00 61 00 64 00 6d 00 69 00 6e 00 69 00 73 00 74 .a.d.m.i.n.i.s.t
00000080 00 72 00 61 00 74 00 6f 00 72 00 00 00 16 00 00 .r.a.t.o.r......
00000090 00 54 00 53 00 2d 00 53 00 54 00 52 00 45 00 53 .T.S.-.S.T.R.E.S
000000a0 00 53 00 31 00 00 00 78 00 00 00 02 00 00 80 44 .S.1...x.......D
000000b0 53 48 4c 02 10 f3 e3 bf b1 37 95 28 80 b7 56 f3 SHL......7.(..V.
000000c0 7c 27 4a 43 cc 50 98 59 05 b5 6b 50 97 62 f8 cf |'JC.P.Y..kP.b..
000000d0 c0 1b 6a 06 16 db b9 b1 ba 21 01 f4 ea 82 dc 37 ..j......!.....7
000000e0 17 65 7d be 58 ec 34 e9 33 07 12 c1 76 8d f5 bc .e}.X.4.3...v...
000000f0 a2 9f 2c ef 32 a7 a4 80 a9 05 f7 02 94 96 8d 95 ..,.2...........
00000100 b8 2c db 55 4a 78 08 eb 87 10 c7 8b a9 0a e6 44 .,.UJx.........D
00000110 ab ec 6b ee 42 bb 32 e7 b0 ef 3c ae 45 73 a6 69 ..k.B.2...<.Es.i
00000120 69 00 00 5a 00 00 00 6a 00 69 00 61 00 7a 00 6f i..Z...j.i.a.z.o
00000130 00 75 00 2d 00 74 00 65 00 73 00 74 00 32 00 2e .u.-.t.e.s.t.2..
00000140 00 74 00 73 00 2d 00 73 00 74 00 72 00 65 00 73 .t.s.-.s.t.r.e.s
00000150 00 73 00 31 00 2e 00 6e 00 74 00 74 00 65 00 73 .s.1...n.t.t.e.s
00000160 00 74 00 2e 00 6d 00 69 00 63 00 72 00 6f 00 73 .t...m.i.c.r.o.s
00000170 00 6f 00 66 00 74 00 2e 00 63 00 6f 00 6d 00 00 .o.f.t...c.o.m..
00000180 00 1a 00 00 00 4a 00 49 00 41 00 5a 00 4f 00 55 .....J.I.A.Z.O.U
00000190 00 2d 00 54 00 45 00 53 00 54 00 32 00 00 00 70 .-.T.E.S.T.2...p
000001a0 00 00 00 02 00 00 00 46 00 00 00 32 00 30 00 30 .......F...2.0.0
000001b0 00 31 00 3a 00 34 00 38 00 39 00 38 00 3a 00 32 .1.:.4.8.9.8.:.2
000001c0 00 62 00 3a 00 32 00 3a 00 39 00 64 00 65 00 37 .b.:.2.:.9.d.e.7
000001d0 00 3a 00 34 00 35 00 36 00 39 00 3a 00 66 00 62 .:.4.5.6.9.:.f.b
000001e0 00 33 00 39 00 3a 00 65 00 66 00 32 00 39 00 00 .3.9.:.e.f.2.9..
000001f0 00 1e 00 00 00 31 00 35 00 37 00 2e 00 35 00 39 .....1.5.7...5.9
00000200 00 2e 00 32 00 34 00 30 00 2e 00 31 00 34 00 34 ...2.4.0...1.4.4
00000210 00 00 00 c0 c0 c0 c0 c0 c0 c0 c0                ...........

03 00 02 1c -> TPKT Header (length = 540 bytes)
02 f0 80 -> X.224 Data TPDU

68 00 01 03 eb 70 82 0d -> PER encoded (basic aligned variant) SendDataIndication
initiator = 1002 (0x03ea)
channelId = 1003 (0x03eb)
dataPriority = high
segmentation = begin | end
userData length = 0x20d = 525 bytes

0d 02 -> TS SHARECONTROLHEADER::totalLength = 0x020d = 525 bytes
0a 00 -> TS_SHARECONTROLHEADER::pduType = 0x000a = PDUTYPE_SERVER_REDIR_PKT (10)
ea 03 -> TS_SHARECONTROLHEADER::pduSource = 0x03ea (1002)

5f 59 -> Padding

00 04 -> RDP_SERVER_REDIRECTION_PACKET::Flags = 0x0400 = SEC_REDIRECTION_PKT
04 02 -> RDP_SERVER_REDIRECTION_PACKET::Length = 0x204 = 516 bytes
```

```
02 00 00 00 -> RDP SERVER REDIRECTION PACKET::SessionID = 2

1d 0b 00 00 -> RDP_SERVER_REDIRECTION_PACKET::RedirFlags = 0x00000b1d
0x00000b1d
= 0x00000800 |
  0x00000200 |
  0x00000100 |
  0x00000010 |
  0x00000008 |
  0x00000004 |
  0x00000001
= LB TARGET NET ADDRESSES |
  LB TARGET NETBIOS NAME |
  LB_TARGET_FQDN |
  LB_PASSWORD |
  LB DOMAIN |
  LB USERNAME |
  LB TARGET NET ADDRESS

46 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetAddressLength = 0x46 = 70 bytes

32 00 30 00 30 00 31 00 3a 00 34 00 38 00 39 00
38 00 3a 00 32 00 62 00 3a 00 32 00 3a 00 39 00
64 00 65 00 37 00 3a 00 34 00 35 00 36 00 39 00
3a 00 66 00 62 00 33 00 39 00 3a 00 65 00 66 00
32 00 39 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetAddress =
"2001:4898:2b:2:9de7:4569:fb39:ef29"

1c 00 00 00 -> RDP SERVER REDIRECTION PACKET::UserNameLength = 0x1c = 28

61 00 64 00 6d 00 69 00 6e 00 69 00 73 00 74 00
72 00 61 00 74 00 6f 00 72 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::UserName =
"administrator"

16 00 00 00 -> RDP SERVER REDIRECTION PACKET::DomainLength = 0x16 = 22 bytes

54 00 53 00 2d 00 53 00 54 00 52 00 45 00 53 00
53 00 31 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::Domain = "TS-STRESS1"

78 00 00 00 -> RDP SERVER REDIRECTION PACKET::PasswordLength = 0x78 = 120 bytes

02 00 00 80 44 53 48 4c 02 10 f3 e3 bf b1 37 95
28 80 b7 56 f3 7c 27 4a 43 cc 50 98 59 05 b5 6b
50 97 62 f8 cf c0 1b 6a 06 16 db b9 b1 ba 21 01
f4 ea 82 dc 37 17 65 7d be 58 ec 34 e9 33 07 12
c1 76 8d f5 bc a2 9f 2c ef 32 a7 a4 80 a9 05 f7
02 94 96 8d 95 b8 2c db 55 4a 78 08 eb 87 10 c7
8b a9 0a e6 44 ab ec 6b ee 42 bb 32 e7 b0 ef 3c
ae 45 73 a6 69 69 00 00 -> RDP_SERVER_REDIRECTION_PACKET::Password

5a 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetFQDNLength = 0x5a = 90

6a 00 69 00 61 00 7a 00 6f 00 75 00 2d 00 74 00
65 00 73 00 74 00 32 00 2e 00 74 00 73 00 2d 00
73 00 74 00 72 00 65 00 73 00 73 00 31 00 2e 00
6e 00 74 00 74 00 65 00 73 00 74 00 2e 00 6d 00
69 00 63 00 72 00 6f 00 73 00 6f 00 66 00 74 00
2e 00 63 00 6f 00 6d 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetFQDN = "jiazou-
test2.ts-stress1.nttest.microsoft.com"

1a 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetNetBiosNameLength = 0x1a = 26

4a 00 49 00 41 00 5a 00 4f 00 55 00 2d 00 54 00
45 00 53 00 54 00 32 00 00 00 -> RDP SERVER REDIRECTION PACKET::TargetNetBiosName =
"JIAZOU-TEST2"
```

```
70 00 00 00 -> RDP_SERVER_REDIRECTION_PACKET::TargetNetAddressesLength = 112 bytes

02 00 00 00 -> TARGET NET ADDRESSES::addressCount = 2

46 00 00 00 -> TARGET NET ADDRESS::addressLength = 70 bytes

32 00 30 00 30 00 31 00 3a 00 34 00 38 00 39 00
38 00 3a 00 32 00 62 00 3a 00 32 00 3a 00 39 00
64 00 65 00 37 00 3a 00 34 00 35 00 36 00 39 00
3a 00 66 00 62 00 33 00 39 00 3a 00 65 00 66 00
32 00 39 00 00 00 -> TARGET NET ADDRESS::address = "2001:4898:2b:2:9de7:4569:fb39:ef29"

1e 00 00 00 -> TARGET_NET_ADDRESS::addressLength = 30 bytes

31 00 35 00 37 00 2e 00 35 00 39 00 2e 00 32 00
34 00 30 00 2e 00 31 00 34 00 34 00 00 00 -> TARGET NET ADDRESS::address =
"157.59.240.144"

c0 c0 c0 c0 c0 c0 c0 c0 -> Excess padding due to over-allocation bug
```

4.6 NineGrid Examples

The diagram below illustrates how a NineGrid bitmap may be resized when it is rendered. The NineGrid Bitmap Information (section 2.2.2.3.1.3.4.1) structure defines the grid layout, specifically the sizes of the LeftWidth, RightWidth, TopHeight, and BottomHeight constants.

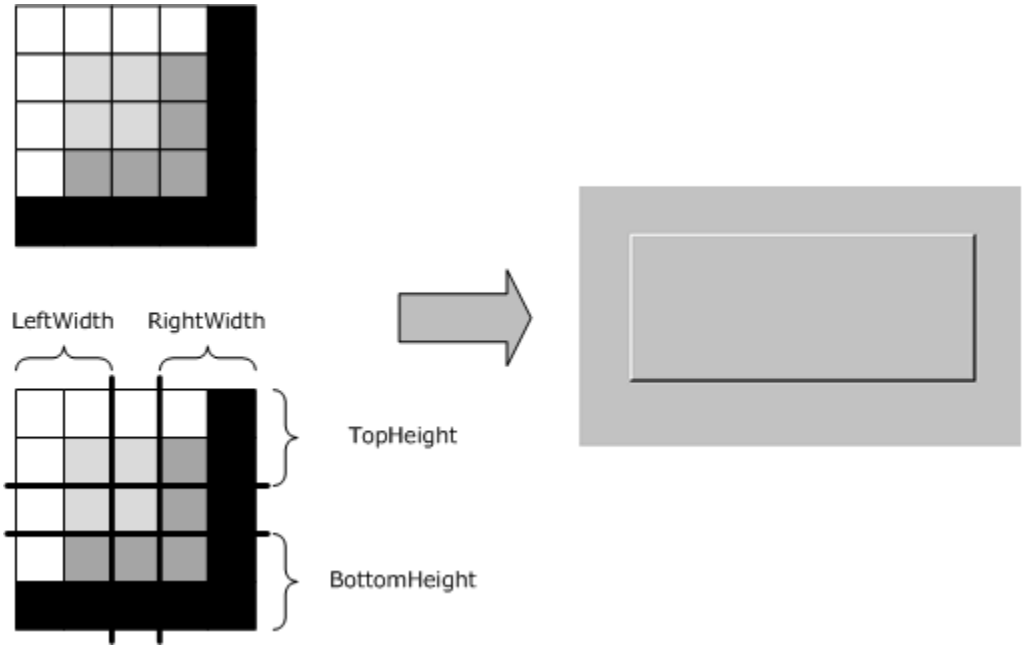


Figure 15: Expansion of a NineGrid bitmap

The diagram below illustrates how a source bitmap in the NineGrid Bitmap Cache (section 3.1.1.1.6) is clipped and then modified using a NineGrid transformation to produce a final image in a destination rectangle.

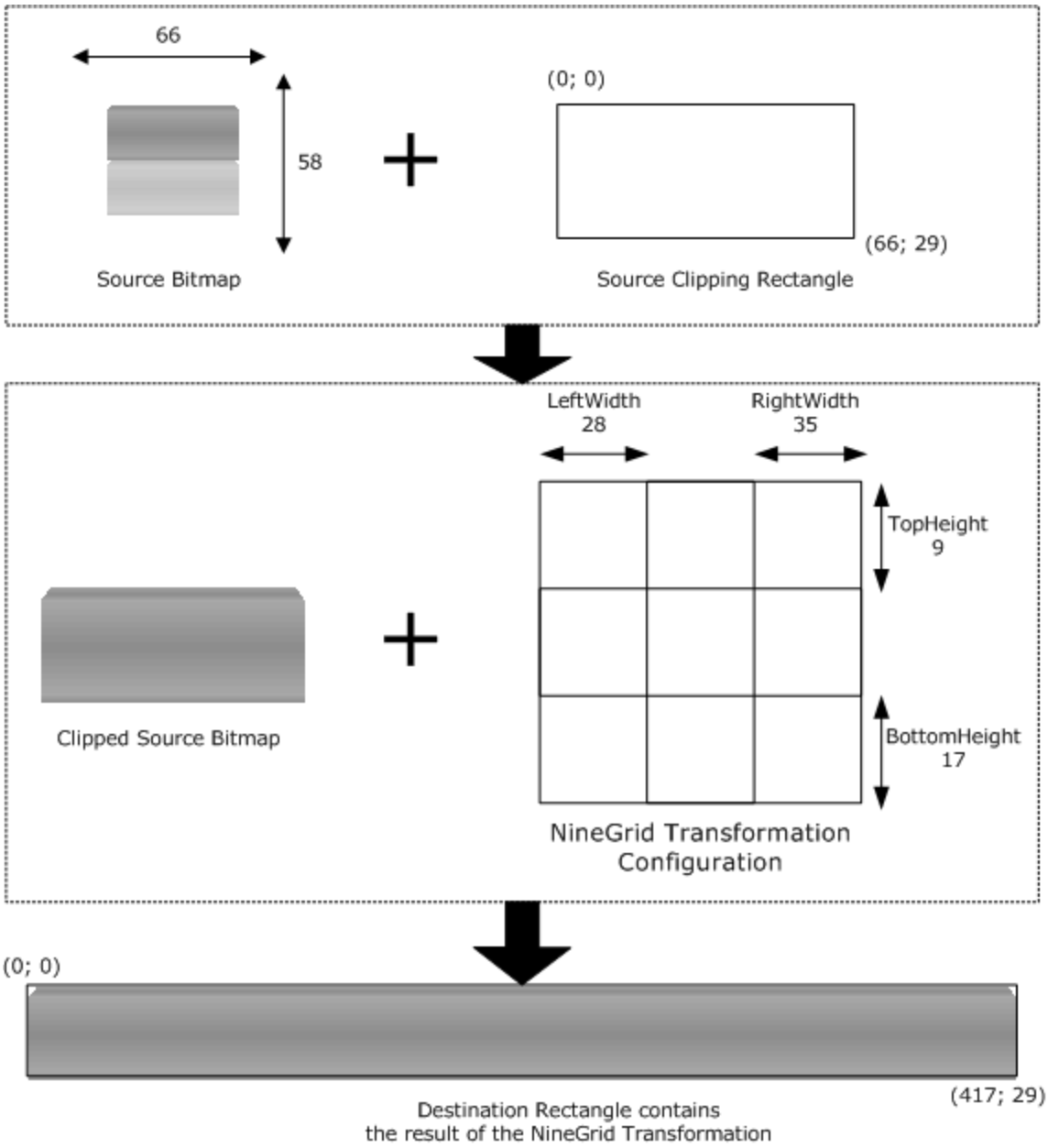


Figure 16: Illustration of Draw Nine Grid Primary Drawing Order

The DrawNineGrid (section 2.2.2.3.1.1.2.21) and MultiDrawNineGrid (section 2.2.2.3.1.1.2.22) Primary Drawing Orders are used to render NineGrid bitmaps that have been stored in the NineGrid Bitmap Cache using the Create NineGrid Bitmap (section 2.2.2.3.1.3.4) and Stream Bitmap Orders (section 2.2.2.3.1.3.5).

4.7 Save Bitmap Example

The SaveBitmap (section 2.2.2.3.1.1.2.12) Primary Drawing Order is used to encode a rectangular screen image for saving or restoring by the client.

The diagram below shows how a 127 pixel by 134 pixel virtual desktop rectangle received in the SaveBitmap Primary Drawing Order is tiled into the client-side Save Bitmap (section 3.2.1.2), and how the position of the next bitmap can be computed.

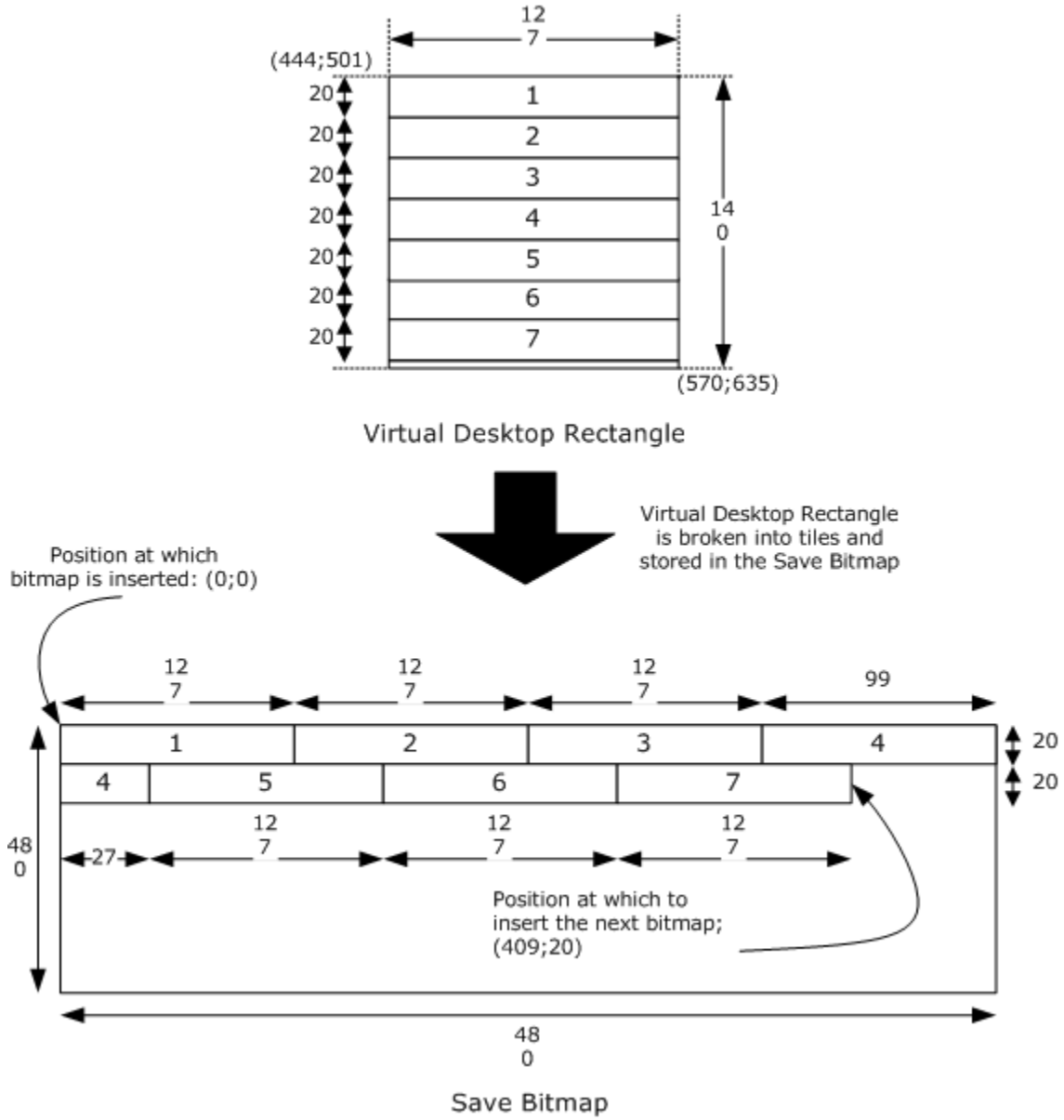


Figure 17: Illustration of Save Bitmap Primary Drawing Order

Applying the formulas in section 2.2.2.3.1.1.2.12 (assuming an X granularity of 1 and a Y granularity of 20):

```
AreaWidthInPixels
= [(width + XGranularity - 1) / XGranularity] * XGranularity
= [(127 + 1 - 1) / 1] * 1
= 127
```

```
AreaHeightInPixels
= [(height + YGranularity - 1) / YGranularity] * YGranularity
= [(134 + 20 - 1) / 20] * 20
= 140

Area
= AreaWidthInPixels * AreaHeightInPixels
= 127 * 140
= 17780
```

Hence, the area occupied by the 127 pixel by 134 pixel bitmap in the SaveBitmap is 17,780 pixels2. The X and Y position in the Save Bitmap from which to tile the next bitmap is computed using the formulas in section 2.2.2.3.1.1.2.12:

```
Y
= [SaveBitmapPosition / (480 * YGranularity)] * YGranularity
= [17780 / (480 * 20)] * 20
= 20

X
= [SaveBitmapPosition - (Y * 480)] / YGranularity
= [17780 - (20 * 480)] / 20
= 409
```

4.8 Glyph Image Data

4.8.1 "d" Character

Glyph image data (1 bpp format) for character "d" extracted from a Cache Glyph (Revision 2) (section 2.2.2.3.1.2.6) Secondary Drawing Order:

```
Glyph width = 5 pixels
Glyph height = 9 pixels
Glyph origin = (0, -9), marked with an "X" on the image grid
Bitmap = { 0x08, 0x08, 0x08, 0x78, 0x88, 0x88, 0x88, 0x88, 0x78 }
```

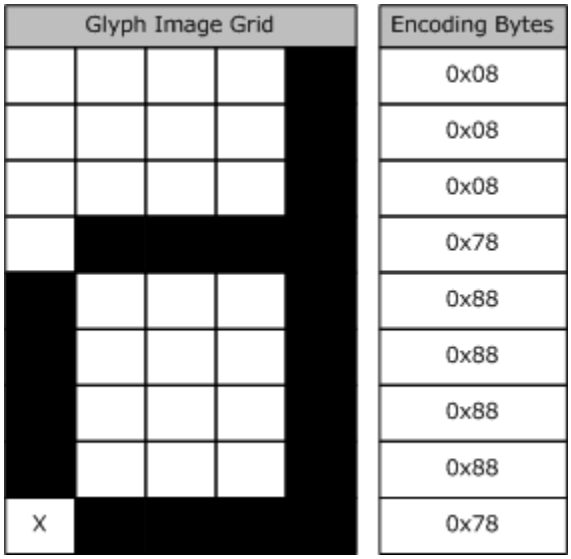


Figure 18: Decoded glyph bytes for "d" character

4.8.2 "p" Character

Glyph image data (1 bpp format) for character "p" extracted from a Cache Glyph (Revision 2) (section 2.2.2.3.1.2.6) Secondary Drawing Order:

```
Glyph width = 5 pixels
Glyph height = 8 pixels
Glyph origin = (0, -6), marked with an "X" on the image grid
Bitmap = { 0xF0, 0x88, 0x88, 0x88, 0x88, 0xF0, 0x80, 0x80 }
```

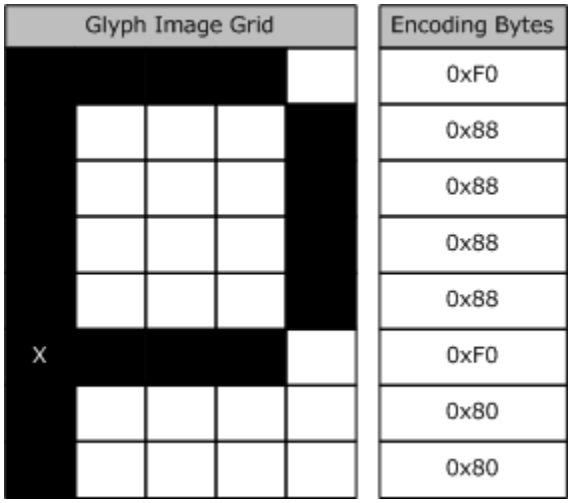


Figure 19: Decoded glyph bytes for "p" character

5 Security

The following sections specify security considerations for implementers of the Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions.

5.1 Security Considerations for Implementers

There are no security considerations for Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions as all traffic is secured by the underlying RDP core protocol. For an overview of the implemented security-related mechanisms, see [MS-RDPBCGR] section 5.

5.2 Index of Security Parameters

None.

6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Vista
- Windows Server 2003
- Windows XP
- Windows 2000
- Windows NT

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.2.2.3.1.1.1.4: The Windows implementation of Remote Desktop Protocol: GDI Acceleration Extension places the field that contains the number of Delta-Encoded Points immediately before order encoding packets containing a DELTA_PTS_FIELD structure.

<2> Section 2.2.2.3.1.1.2.15: This Unicode value is only for debugging purposes and is not always accurate in Microsoft RDP implementations.

<3> Section 2.2.2.3.1.1.2.18: The Windows implementation of RDP set this field to 0x0000.

<4> Section 2.2.2.3.1.2.7: The Windows implementation of Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions sets this field to 0x00.

<5> Section 2.2.2.3.1.3.6.5: The Windows implementation of Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions sets this field to 0x00.

<6> Section 2.2.2.3.1.3.6.6: The Windows implementation of Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions sets this field to 0x00.

<7> Section 2.2.2.3.1.3.6.7: The Windows implementation of Remote Desktop Protocol: Graphics Device Interface (GDI) Acceleration Extensions sets this field to 0x00.

7 Index

A

Abstract data model
  client (section 3.1.1, section 3.2.1)
  RDP 6.0-based bulk data compression
  server (section 3.1.1, section 3.3.1)
Accelerated graphics
  caches
  drawing orders (section 1.3.1.2, section 2.2.2.2.1)
  error conditions (section 1.3.1.3, section 2.2.2.4)
  overview (section 1.3.1, section 2.2.2)
Alternate secondary drawing orders (section 1.3.1.2.3, section 2.2.2.3.1.3, section 3.2.5.1.3, section 3.3.5.1.3)
ALTSEC_DRAWING_ORDER_HEADER packet
Annotated alternate secondary drawing orders examples
Annotated primary drawing orders examples
Annotated secondary drawing orders examples
Applicability

B

Bitmap
  compressing
  decompressing
  Save Bitmap
Bitmap Cache Wait List
Bitmap caches
Bitmap keys - persistent
Brush caches

C

CACHE_BITMAP_ORDER packet
CACHE_BITMAP_REV2_ORDER packet
CACHE_BRUSH_ORDER packet
CACHE_COLOR_TABLE_ORDER packet
CACHE_GLYPH_ORDER packet
CACHE_GLYPH_REV2_ORDER packet
Caches
  bitmap caches
  brush caches
  color table caches
  fragment caches
  GDI+ caches
  glyph caches
  NineGrid bitmap caches
  offscreen bitmap caches
  overview
Capability negotiation
Capability sets
Client
  abstract data model (section 3.1.1, section 3.2.1)
  alternate secondary drawing orders
  drawing orders
  higher-layer triggered events (section 3.1.4, section 3.2.4)
  initialization (section 3.1.3, section 3.2.3)
  local events (section 3.1.7, section 3.2.7)
  message processing (section 3.1.5, section 3.2.5)
  overview (section 3.1, section 3.2)
  primary drawing orders
  RDP 6.0 bitmap compression
  RDP 6.0-based bulk data compression
  secondary drawing orders
  sequencing rules (section 3.1.5, section 3.2.5)
  timer events (section 3.1.6, section 3.2.6)
  timers (section 3.1.2, section 3.2.2)
Client Bitmap Cache Error PDU packet
Client DrawNineGrid Cache Error PDU packet
Client GDI+ Error PDU packet
Color table caches
Compressed stream - decoding
COMPRESSED_COLOR_BRUSH packet
COORD_FIELD packet
Copy-offset tables
CREATE_NINEGRID_BITMAP_ORDER packet
CREATE_OFFSCR_BITMAP_ORDER packet

D

Data
  compressing - RDP 6.0-based bulk data compression
  decompressing - RDP 6.0-based bulk data compression
Data model - abstract
  client (section 3.1.1, section 3.2.1)
  RDP 6.0-based bulk data compression
  server (section 3.1.1, section 3.3.1)
DELTA_PTS_FIELD packet
DELTA_RECTS_FIELD packet
DRAW_GDIPLUS_CACHE_END_ORDER packet
DRAW_GDIPLUS_CACHE_FIRST_ORDER packet
DRAW_GDIPLUS_CACHE_NEXT_ORDER packet
DRAW_GDIPLUS_CACHE_TYPE packet
DRAW_GDIPLUS_END_ORDER packet
DRAW_GDIPLUS_FIRST_ORDER packet
DRAW_GDIPLUS_NEXT_ORDER packet
Drawing orders
  alternate secondary (section 1.3.1.2.3, section 2.2.2.3.1.3)
  alternate secondary drawing orders (section 3.2.5.1.3, section 3.3.5.1.3)
  client
  overview (section 1.3.1.2, section 2.2.2.2.1)
  primary (section 1.3.1.2.1, section 2.2.2.3.1.1)
  primary drawing orders (section 3.2.5.1.1, section 3.3.5.1.1)
  secondary (section 1.3.1.2.2, section 2.2.2.3.1.2)
  secondary drawing orders (section 3.2.5.1.2, section 3.3.5.1.2)
  server
DRAWING_ORDER packet
DRAWNINEGRID_ORDER packet
DSTBLT_ORDER packet

E

ELLIPSE_CB_ORDER packet
ELLIPSE_SC_ORDER packet
Enhanced RDP security
Enhanced security server redirection example
Enhanced Security Server Redirection PDU packet
EOS tables
Error conditions (section 1.3.1.3, section 2.2.2.4, section 3.2.5.2, section 3.3.5.2)
Examples
  annotated alternate secondary drawing orders examples
  annotated primary drawing orders examples
  annotated secondary drawing orders examples
  enhanced security server redirection example
  glyph image example
  NineGrid example
  overview
  SaveBitmap example
  standard security server redirection example

F

FASTGLYPH_ORDER packet
FASTINDEX_ORDER packet
Fields - vendor-extensible
FILL_MODE packet
FOUR_BYTE_UNSIGNED_ENCODING packet
Fragment caches

G

GDI+ caches
Glossary
Glyph caches
Glyph image example
GLYPHINDEX_ORDER packet

H

Higher-layer triggered events
  client (section 3.1.4, section 3.2.4)
  server (section 3.1.4, section 3.3.4)

I

Implementer - security considerations
Index of security parameters
Informative references
Initialization
  client (section 3.1.3, section 3.2.3)
  server (section 3.1.3, section 3.3.3)

L

Length-of-match tables
LINETO_ORDER packet
Literal tables
Local events
  client (section 3.1.7, section 3.2.7)
  server (section 3.1.7, section 3.3.7)
Logically-compressed stream - encoding

M

MEM3BLT_ORDER packet
MEMBLT_ORDER packet
Message processing
  client (section 3.1.5, section 3.2.5)
  server (section 3.1.5, section 3.3.5)
Messages
  overview
  syntax
  transport
MULTI_DRAWNINEGRID_ORDER packet
MULTI_DSTBLT_ORDER packet
MULTI_OPAQUERECT_ORDER packet
MULTI_PATBLT_ORDER packet
MULTI_SCRBLT_ORDER packet

N

NineGrid bitmap caches
NineGrid example
NINEGRID_BITMAP_INFO packet
Normative references

O

OFFSCR_DELETE_LIST packet
Offscreen Bitmap Cache Error PDU packet
Offscreen bitmap caches
OPAQUERECT_ORDER packet
Overview (synopsis)

P

Parameters - security index
PATBLT_ORDER packet
Persistent bitmap keys
POLYGON_CB_ORDER packet
POLYGON_SC_ORDER packet
POLYLINE_ORDER packet
Preconditions
Prerequisites
Primary drawing order history (section 3.2.1.1, section 3.3.1.2)
Primary drawing orders (section 1.3.1.2.1, section 2.2.2.3.1.1, section 3.2.5.1.1, section 3.3.5.1.1)
PRIMARY_DRAWING_ORDER packet

R

RDP 6.0 bitmap compression
  client
  compressing bitmaps
  decompressing bitmaps
  run-length encoding
  server
  techniques
RDP 6.0 bulk compression

RDP 6.0-based bulk data compression
abstract data model
client
compressing data
decompressing data
server
wire format
RDP security
enhanced
standard
RDP_SERVER_REDIRECTION_PACKET packet
RDP6_BITMAP_STREAM packet
RDP6_RLE_SEGMENT packet
RDP6_RLE_SEGMENTS packet
References
informative
normative
overview
Relationship to other protocols
ROP2_OPERATION packet
ROP3_OPERATION_INDEX packet
Run-length encoding

S

Save Bitmap
SaveBitmap example
SAVEBITMAP_ORDER packet
SCRBLT_ORDER packet
Secondary drawing orders (section 1.3.1.2.2, section 2.2.2.3.1.2, section 3.2.5.1.2, section 3.3.5.1.2)
SECONDARY_DRAWING_ORDER_HEADER packet
Security
enhanced RDP security
implementer considerations
overview
parameter index
standard RDP security
Sequencing rules
client (section 3.1.5, section 3.2.5)
server (section 3.1.5, section 3.3.5)
Server
abstract data model (section 3.1.1, section 3.3.1)
alternate secondary drawing orders
drawing orders
higher-layer triggered events (section 3.1.4, section 3.3.4)
initialization (section 3.1.3, section 3.3.3)
local events (section 3.1.7, section 3.3.7)
message processing (section 3.1.5, section 3.3.5)
overview (section 3.1, section 3.3)
primary drawing orders
RDP 6.0 bitmap compression
RDP 6.0-based bulk data compression
redirection
secondary drawing orders
sequencing rules (section 3.1.5, section 3.3.5)
timer events (section 3.1.6, section 3.3.6)
timers (section 3.1.2, section 3.3.2)
Server redirection (section 1.3.3, section 2.2.3)
Server redirection PDU
processing
sending
Standard RDP security
Standard security server redirection example
Standard Security Server Redirection PDU packet
Standards assignments
STREAM_BITMAP_FIRST_ORDER packet
STREAM_BITMAP_NEXT_ORDER packet
SWITCH_SURFACE_ORDER packet
Syntax - message

T

TARGET_NET_ADDRESS packet
TARGET_NET_ADDRESSES packet
Timer events
client (section 3.1.6, section 3.2.6)
server (section 3.1.6, section 3.3.6)
Timers
client (section 3.1.2, section 3.2.2)
server (section 3.1.2, section 3.3.2)
Transport - message
Triggered events - higher-layer
client (section 3.1.4, section 3.2.4)
server (section 3.1.4, section 3.3.4)
TS_BITMAP_CACHE_ERROR_INFO packet
TS_BITMAP_CACHE_ERROR_PDU packet
TS_CACHE_GLYPH_DATA packet
TS_CACHE_GLYPH_DATA_REV2 packet
TS_COLOR packet
TS_COLOR_QUAD packet
TS_COLORREF packet
TS_COLORTABLE_CAPABILITYSET packet
TS_DRAW_GDIPLUS_CAPABILITYSET packet
TS_DRAW_NINEGRID_CAPABILITYSET packet
TS_DRAWGDIPLUS_ERROR_PDU packet
TS_DRAWNINEGRID_ERROR_PDU packet
TS_FP_UPDATE_ORDERS packet
TS_GDIPLUS_CACHE_CHUNK_SIZE packet
TS_GDIPLUS_CACHE_ENTRIES packet
TS_GDIPLUS_IMAGE_CACHE_PROPERTIES packet
TS_OFFSCRCACHE_ERROR_PDU packet
TS_UPDATE_ORDERS_PDU_DATA packet
TWO_BYTE_SIGNED_ENCODING packet
TWO_BYTE_UNSIGNED_ENCODING packet

V

VARIABLE1_FIELD packet
VARIABLE2_FIELD packet
Vendor-extensible fields
Versioning

W

Windows behavior
Wire format - RDP 6.0-based bulk data compression
decoding compressed stream
encoding logically compressed stream
length-of-match tables
literal and EOS and copy-offset tables

overview