Direct Print Protocol Specification

Version 1.0
September 15, 1998

Sponsored by:
Digital Still Image Working Group of 1394 Trade Association

Approved for Released by:
1394 Trade Association Board of Directors
TABLE OF CONTENTS

1. SCOPE AND PURPOSE 1

2. DEFINITIONS AND ABBREVIATIONS 2
   2.1. Conformance Glossary 2
   2.2. Technical Glossary 2
      2.2.1. Section 5 / Thin Protocol 2
      2.2.2. Section 6 / Direct Print Command Set 2

3. BIT, BYTE AND QUADLET ORDERING 3
   3.1. Bit ordering within a byte 3
   3.2. Byte ordering within a quadlet 3
   3.3. Quadlet ordering within an octlet 3

4. DIRECT PRINT MODEL 4

5. THIN PROTOCOL 6
   5.1. Overview 6
   5.2. Service Model 6
   5.3. Thin Protocol Layers 7
      5.3.1. Thin Session Layer 7
      5.3.2. Thin Transaction Layer 7
      5.3.3. Thin Session Service 7
      5.3.4. Thin Transaction Service 8
   5.4. Thin Protocol Sequence 9
   5.5. Thin Protocol Segmentation 11
   5.6. Register Space 12
5.6.1. Connection Register  
5.6.2. SDU Management Register  
5.6.3. SDU Register  
5.6.4. Register Space Summary  

5.7. Connection Information

5.8. Connection Sequence  
5.8.1. Connect  
5.8.1.1. Dynamic Register Allocation Algorithm  
5.8.2. Reconnect  
5.8.2.1. Reconnect sequence  
5.8.3. Disconnect  

5.9. Asynchronous Transfer  
5.9.1. Thin Protocol Data Transfer  
5.9.1.1. Thin Session Single Data Transfer Sequence  
5.9.1.2. Thin Session Segment Data Transfer Sequence  
5.9.1.3. Thin Transaction Data Transfer Sequence  
5.9.1.4. Thin Transaction SDU Transfer Stop Sequence  
5.9.2. Application Command Transfer  
5.9.3. Multiple Data Transfer  
5.9.4. Bi-directional Data Transfer  
5.9.5. Abort  

5.10. Timeout Sequence  
5.10.1. Connect Timeout  
5.10.2. Reconnect Timeout  
5.10.3. Idle Timeout  
5.10.4. Data Transfer Timeout  

5.11. Thin Protocol Format  
5.11.1. Thin Session Service  
5.11.2. Thin Session Field Format  
5.11.3. Negotiation Information  
5.11.3.1. Negotiation Information Tag  
5.11.3.2. Negotiation Information Format  
5.11.4. Format used for Connection
5.11.4.1. Connect Request Format
5.11.4.2. Connect Ack Format
5.11.4.3. Connect Nack Format
5.11.5. Format used for Reconnection
  5.11.5.1. Reconnect Request Format
  5.11.5.2. Reconnect Ack Format
  5.11.5.3. Reconnect Nack Format
5.11.6. Format used for Disconnection
  5.11.6.1. Disconnect Format
5.11.7. Format Used for Data Transfer
  5.11.7.1. Application Data Segmentation
  5.11.7.2. Single Data Format
  5.11.7.3. Segment Data Format
5.11.8. Format used for Command Abort
  5.11.8.1. Abort Format
5.11.9. Thin Transaction Service
5.11.10. Register Space
5.11.11. Data Specification Field
5.11.12. Operation used for Connection
  5.11.12.1. Connect Request
  5.11.12.2. Connect Response
5.11.13. Operation used for Reconnection
  5.11.13.1. Reconnect Request
  5.11.13.2. Reconnect Response
5.11.14. Operation used for Disconnection
  5.11.14.1. Disconnect Request
5.11.15. Operation used for SDU Transfer
  5.11.15.1. SDU Transfer
  5.11.15.2. SDU Transfer Completion
  5.11.15.3. SDU Transfer Stop
  5.11.15.4. SDU Transfer Ready
  5.11.15.5. SDU Transfer Reject

6. DIRECT PRINT COMMAND SET

6.1. Overview
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.1.</td>
<td>Command Model</td>
<td>72</td>
</tr>
<tr>
<td>6.1.2.</td>
<td>Command Sequence</td>
<td>73</td>
</tr>
<tr>
<td>6.1.3.</td>
<td>Negotiation</td>
<td>74</td>
</tr>
<tr>
<td>6.2.</td>
<td>Requirements</td>
<td>74</td>
</tr>
<tr>
<td>6.3.</td>
<td>Command Set Categorization</td>
<td>75</td>
</tr>
<tr>
<td>6.3.1.</td>
<td>Command Set Components</td>
<td>75</td>
</tr>
<tr>
<td>6.3.2.</td>
<td>Command Categorization</td>
<td>75</td>
</tr>
<tr>
<td>6.3.3.</td>
<td>Command List</td>
<td>77</td>
</tr>
<tr>
<td>6.4.</td>
<td>Negotiation Command</td>
<td>78</td>
</tr>
<tr>
<td>6.4.1.</td>
<td>Implementation Level in Negotiation Command</td>
<td>79</td>
</tr>
<tr>
<td>6.4.1.1.</td>
<td>Implementation Level of Negotiation Command</td>
<td>79</td>
</tr>
<tr>
<td>6.4.1.2.</td>
<td>Implementation Level of Item</td>
<td>80</td>
</tr>
<tr>
<td>6.4.1.3.</td>
<td>Implementation Level of Value</td>
<td>81</td>
</tr>
<tr>
<td>6.4.2.</td>
<td>Item/Value Listing</td>
<td>82</td>
</tr>
<tr>
<td>6.4.3.</td>
<td>Negotiation Command/Response Packet Format</td>
<td>84</td>
</tr>
<tr>
<td>6.4.4.</td>
<td>Query Unit List Format</td>
<td>86</td>
</tr>
<tr>
<td>6.4.4.1.</td>
<td>Item Unit Format</td>
<td>88</td>
</tr>
<tr>
<td>6.4.4.2.</td>
<td>Value Content Block Format</td>
<td>91</td>
</tr>
<tr>
<td>6.4.5.</td>
<td>Negotiation Command Details</td>
<td>94</td>
</tr>
<tr>
<td>6.4.5.1.</td>
<td>GetQueryItem</td>
<td>95</td>
</tr>
<tr>
<td>6.4.5.2.</td>
<td>SetQueryItem</td>
<td>96</td>
</tr>
<tr>
<td>6.4.6.</td>
<td>Negotiation Item/Value Details</td>
<td>97</td>
</tr>
<tr>
<td>6.4.6.1.</td>
<td>VendorUniqueItem</td>
<td>98</td>
</tr>
<tr>
<td>6.4.6.2.</td>
<td>ImageFormat</td>
<td>99</td>
</tr>
<tr>
<td>6.4.6.3.</td>
<td>ImageSize</td>
<td>100</td>
</tr>
<tr>
<td>6.4.6.4.</td>
<td>OutputOrientation</td>
<td>102</td>
</tr>
<tr>
<td>6.4.6.5.</td>
<td>Sizing</td>
<td>103</td>
</tr>
<tr>
<td>6.4.6.6.</td>
<td>PosX</td>
<td>104</td>
</tr>
<tr>
<td>6.4.6.7.</td>
<td>PosY</td>
<td>105</td>
</tr>
<tr>
<td>6.4.6.8.</td>
<td>NumPics</td>
<td>106</td>
</tr>
<tr>
<td>6.4.6.9.</td>
<td>NumCopies</td>
<td>107</td>
</tr>
<tr>
<td>6.5.</td>
<td>Generic Command</td>
<td>108</td>
</tr>
<tr>
<td>6.5.1.</td>
<td>Implementation Level in Generic Command</td>
<td>109</td>
</tr>
</tbody>
</table>
6.5.2. Generic Command/Response Packet Format 110
6.5.3. Generic Command/Response Parameter Format 112
6.5.4. Generic Command Details 114
  6.5.4.1. Send 115
  6.5.4.2. Cancel 116
  6.5.4.3. GetStatus 117
  6.5.4.4. SendStatus 118

6.6. Vendor Unique Command 119
  6.6.1. Vendor Unique Command/Response Packet Format 119

6.7. Command Sequence 120
  6.7.1. Command Sequence Overview 120
  6.7.2. Negotiation 121
    6.7.2.1. Commands for Negotiation 121
    6.7.2.2. Negotiation Sequence 122
  6.7.3. Data Transfer 125
    6.7.3.1. One Picture Per One Output Unit 125
    6.7.3.2. Multiple Pictures Per One Output Unit 126
  6.7.4. Others 127
    6.7.4.1. Cancel command 127
    6.7.4.2. GetStatus command 127
    6.7.4.3. SendStatus command 127

7. CONFIGURATION ROM 128

7.1. Configuration ROM Definition for THIN PROTOCOL 129
  7.1.1. Power reset initialization 129
  7.1.2. Bus information block 130
  7.1.3. Root directory 132
    7.1.3.1. Module_Vendor_ID entry 132
    7.1.3.2. Node_Capabilities entry 132
    7.1.3.3. Unit_Directory entry 133
  7.1.4. Unit directory 133
    7.1.4.1. Unit_Spec_ID entry 134
    7.1.4.2. Unit_SW_Version entry 134
    7.1.4.3. Command_Set_Spec_ID entry 135
7.1.4.4. Command_Set entry 135
7.1.4.5. Command_Set_Details entry 136
7.1.4.6. Connection_Register entry 136
7.1.4.7. Write Transaction Interval entry (optional) 136

7.2. Configuration ROM Definition for Direct Print Command Set 138
7.2.1. Command_Set_Spec_ID entry 138
7.2.2. Command_Set entry 138
7.2.3. Command_Set_Details entry 139

7.3. Instance Directory 140
7.3.1. Root Directory 140
7.3.1.1. Instance_Directory entry 140
7.3.2. Instance Directory 141
7.3.2.1. Function_Class entry 141
7.3.2.2. Unit_Directory entry 141

ANNEX A. DPP IMPLEMENTATION EXAMPLES 142

ANNEX B. NEGOTIATION IMPLEMENTATION 148

ANNEX C. IMAGE DATA FORMAT DETAILS 150

ANNEX D. SAMPLE CONFIGURATION ROM 156

ANNEX E. ENUMERATION OF DPP DEVICES USING CONFIGURATION ROM 161

ANNEX F. THIN PROTOCOL STATE MACHINE (INFORMATIVE) 164
LIST OF FIGURES

Figure 1  Bit ordering ........................................................................................................... 3
Figure 2  Byte ordering ....................................................................................................... 3
Figure 3  Quadlet ordering .................................................................................................. 3
Figure 4  Direct Print Model .............................................................................................. 4
Figure 5  Service model ..................................................................................................... 7
Figure 6  Thin protocol sequence ..................................................................................... 10
Figure 7  Thin Protocol segmentation .............................................................................. 11
Figure 8  Register space .................................................................................................... 12
Figure 9  Connect sequence ............................................................................................. 16
Figure 10 Dynamic register allocation sequence - 1 ......................................................... 17
Figure 11 Dynamic register allocation sequence - 2 ......................................................... 18
Figure 12 Dynamic register allocation sequence - 3 ......................................................... 19
Figure 13 Reconnect sequence ......................................................................................... 22
Figure 14 Disconnect sequence ....................................................................................... 24
Figure 15 Thin Protocol segmentation ............................................................................ 25
Figure 16 Thin Session Single data transfer .................................................................... 27
Figure 17 Thin Session Segment data transfer ................................................................ 29
Figure 18 Thin Transaction data transfer ........................................................................ 31
Figure 19 Thin Transaction data transfer stop ................................................................. 32
Figure 20 Application command transfer ......................................................................... 34
Figure 21 Multiple data transfer ..................................................................................... 35
Figure 22 Bi-directional data transfer .............................................................................. 36
Figure 23 Abort sequence ............................................................................................... 38
Figure 24 Connect timeout .............................................................................................. 39
Figure 25 Reconnect timeout ......................................................................................... 40
Figure 26 Idle timeout ..................................................................................................... 41
Figure 27 Data transfer timeout - 1 ................................................................................. 43
Figure 28 Data transfer timeout - 2 ................................................................................. 43
Figure 29 Data transfer timeout - 3 ................................................................................ 44
Figure 30 Thin Session field format ................................................................................. 45
Figure 31 Negotiation information format - 2 ................................................................. 48
Figure 32 Vendor ID ......................................................................................................... 48
Figure 33 Command set ID .............................................................................................. 48
Figure 34 Transfer mode ................................................................................................. 49
Figure 35 Timeout ........................................................................................................... 49
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
<td>Unit_SW_Version entry format</td>
<td>134</td>
</tr>
<tr>
<td>109</td>
<td>Command_Set_Spec_ID entry format</td>
<td>135</td>
</tr>
<tr>
<td>110</td>
<td>Command_Set entry format</td>
<td>135</td>
</tr>
<tr>
<td>111</td>
<td>Command_Set_Details entry format</td>
<td>136</td>
</tr>
<tr>
<td>112</td>
<td>Connection_Register entry format</td>
<td>136</td>
</tr>
<tr>
<td>113</td>
<td>Write_Transaction_Interval entry format</td>
<td>136</td>
</tr>
<tr>
<td>114</td>
<td>Command_Set_Spec_ID entry format</td>
<td>138</td>
</tr>
<tr>
<td>115</td>
<td>Command_Set entry format</td>
<td>138</td>
</tr>
<tr>
<td>116</td>
<td>Command_Set_Details entry format</td>
<td>139</td>
</tr>
<tr>
<td>117</td>
<td>Instance_Directory entry format</td>
<td>140</td>
</tr>
<tr>
<td>118</td>
<td>Function_Class entry format</td>
<td>141</td>
</tr>
<tr>
<td>119</td>
<td>Unit_Directory entry format</td>
<td>141</td>
</tr>
<tr>
<td>120</td>
<td>Total DPP Flow Example</td>
<td>142</td>
</tr>
<tr>
<td>121</td>
<td>An Example of Sequential Query and Set</td>
<td>143</td>
</tr>
<tr>
<td>122</td>
<td>An Example of Multiple Query and Set</td>
<td>144</td>
</tr>
<tr>
<td>123</td>
<td>An Example of GetQueryItem Command</td>
<td>145</td>
</tr>
<tr>
<td>124</td>
<td>An Example of GetQueryItem response</td>
<td>145</td>
</tr>
<tr>
<td>125</td>
<td>An Example of SetQueryItem Command</td>
<td>146</td>
</tr>
<tr>
<td>126</td>
<td>An Example of SetQueryItem Response - 1</td>
<td>146</td>
</tr>
<tr>
<td>127</td>
<td>An Example of SetQueryItem Response – 2</td>
<td>146</td>
</tr>
<tr>
<td>128</td>
<td>An Example of Send Command</td>
<td>147</td>
</tr>
<tr>
<td>129</td>
<td>An Example of Send Response – 1</td>
<td>147</td>
</tr>
<tr>
<td>130</td>
<td>An Example of Send Response - 2</td>
<td>147</td>
</tr>
<tr>
<td>131</td>
<td>Steps of Negotiation in Sequential Query and Set</td>
<td>148</td>
</tr>
<tr>
<td>132</td>
<td>Steps of Negotiation in Multiple Query and Set</td>
<td>149</td>
</tr>
<tr>
<td>133</td>
<td>Categorization of Image Data Format</td>
<td>150</td>
</tr>
<tr>
<td>134</td>
<td>Image Data Arrangement of rawRGB</td>
<td>151</td>
</tr>
<tr>
<td>135</td>
<td>Image Data Arrangement of D-rawRGB</td>
<td>152</td>
</tr>
<tr>
<td>136</td>
<td>Image Data Arrangement of Exif</td>
<td>154</td>
</tr>
<tr>
<td>137</td>
<td>Image Data Arrangement of JFIF</td>
<td>155</td>
</tr>
<tr>
<td>138</td>
<td>Configuration ROM Sample of Basic Device</td>
<td>156</td>
</tr>
<tr>
<td>139</td>
<td>Configuration ROM Sample of Multi-function device</td>
<td>158</td>
</tr>
<tr>
<td>140</td>
<td>Configuration ROM Sample of Multi-Protocol device</td>
<td>160</td>
</tr>
<tr>
<td>141</td>
<td>Multiple Device Topology</td>
<td>161</td>
</tr>
<tr>
<td>142</td>
<td>Step1: Printer Device Discovery</td>
<td>162</td>
</tr>
<tr>
<td>143</td>
<td>Step2: DPP Printer Device Discovery</td>
<td>162</td>
</tr>
</tbody>
</table>
Figure 144  Configuration ROM Information for Device Discovery............................. 163
Figure 145  Thin Session connect state machine.................................................... 164
Figure 146  Thin Session connect state machine (continued) .................................. 165
Figure 147  Thin Session outbound data transfer state machine ............................... 166
Figure 148  Thin Session inbound data transfer state machine ................................. 168
Figure 149  Thin Transaction state machine ............................................................... 169
LIST OF TABLES

Table 1    Register space summary................................................................. 14
Table 2    Connection information................................................................. 14
Table 3    MsgType defined value.................................................................... 46
Table 4    InfType defined value ..................................................................... 46
Table 5    NegInf Tag defined value................................................................. 47
Table 6    Default timeout value..................................................................... 49
Table 7    Result of connect rejection.............................................................. 53
Table 8    Result of reconnect rejection.......................................................... 56
Table 9    Result of disconnect....................................................................... 57
Table 10   Command type .............................................................................. 60
Table 11   Segment flag.................................................................................. 61
Table 12   Data type defined value................................................................. 66
Table 13   Command categorization................................................................. 75
Table 14   Packet Format categorization......................................................... 76
Table 15   Command list ................................................................................ 77
Table 16   Item and Value............................................................................... 79
Table 17   Implementation level of Negotiation Command............................... 79
Table 18   Implementation level of Item........................................................... 80
Table 19   Implementation level of Value.......................................................... 81
Table 20   Item/Value list................................................................................ 82
Table 21   Item Code....................................................................................... 89
Table 22   Implementation level in Generic Command...................................... 109
Table 23   Send Functionality......................................................................... 125
Table 24   DPP defined image data formats.................................................... 150
Table 25   Viewing Environment of sRGB....................................................... 151
Table 26   Additional Restrictions................................................................. 153
Table 27   Exif Tag handling.......................................................................... 153
Table 28   Additional Restrictions................................................................. 155
Revision History
Version 1.0 (September 15, 1998)
1. **Scope and Purpose**

Digital Still Image handling devices such as Digital Still Camera, Digital Camcorder, Scanner and Set Top Box, are evolving rapidly. As a result, they have created new connectivity requirements for Printers to provide output for these Digital Still Image devices.

Such digital still image handling devices usually exchange image data via a host machine such as a PC. "Peer to Peer" connection is one of the advantages of IEEE1394 High Performance Serial Bus. This allows Printers to be connected directly with Digital Image Input Devices to order to print out without host PC machines. It is so called a Direct Print Application. The Direct Print Application should have the ability to enable users to use Printers much more easily and will create a new Digital Still Image culture and market.

This document specifies Direct Print Protocol as a common data transfer protocol to realize Direct Print Application. Direct Print Protocol consists of two parts: Command Set and Data Transfer. This protocol ensures an output of one sheet of print regardless of the combination of image source devices and target devices. It also provides a possibility to obtain better output according to the combinations and capabilities of the devices.
2. Definitions and Abbreviations

2.1. Conformance Glossary

- **may**: A keyword that indicates flexibility of choice with no implied preference.
- **shall**: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products conforming to this standard.
- **should**: A keyword that indicates flexibility of choice with a strongly preferred alternative. This term is equivalent to the phrase “is recommended”

2.2. Technical Glossary

2.2.1. Section 5 / Thin Protocol

- **Application data**: Data created by an Application. All data that an application issues to the Thin Session, are Application data.
- **Application command**: Application data that includes a command to be executed. The commands described in Direct Print Command set are treated as Application commands.
- **Thin Protocol packet**: Packet created by the Thin Protocol.
- **SDU**: Segment Data Unit
- **Machine ID**: The ID of the node. This ID is specified by an Application. In this document, node_unique_id is used as the Machine ID.
- **Initiator node**: Node that issues a connect request.
- **Responder node**: Node that responds to a connect request.
- **Sender node**: Node that issues Application data or a disconnect/reconnect/abort request.
- **Receiver node**: Node that receives Application data or a disconnect/reconnect/abort request.
- **Requester node**: Node that issues a command request.
- **Executor node**: Node that receives a command request and executes the command.

2.2.2. Section 6 / Direct Print Command Set

- **Image source device**: The device such as a digital still camera or scanner that captures or has source image(s) to be sent to the target device.
- **Target device**: The device such as a printer that outputs images from an image source device or that has the behavior equivalent to printing.
- **Command issuing device**: The device sending the command to a command responding device.
- **Command responding device**: The device sending the response to a command issuing device in reply to the received command.
- **Item**: Category of parameter group that is negotiable between an image source device and a target device.
• **Value:** Each parameter of Item representing the concrete capability.

• **Exif:** Exchangeable image file format for Digital Still Camera (JEIDA-49-1997:Digital Still Camera Image File Format Standard, JEIDA.)

• **JFIF:** JPEG File Interchange Format, see “JPEG File Interchange Format version 1.02 Eric Hamilton C-Cube Microsystems Sep1, 1992.”

3. **Bit, Byte and Quadlet ordering**

3.1. **Bit ordering within a byte**
Within a byte, the most significant bit, msb, is that which is transmitted first and the least significant bit, lsb, is that which is transmitted last on the Serial Bus, as shown below.

```
| msb |  |  |  |  |  |  |  |  | lsb |
```

**Figure 1** Bit ordering

3.2. **Byte ordering within a quadlet**
Within a quadlet, the most significant byte is that which is transmitted first and the least significant byte is that which is transmitted last on the Serial Bus, as shown below.

```
| most significant byte | second most significant byte | next to least significant byte | least significant byte |
```

**Figure 2** Byte ordering

3.3. **Quadlet ordering within an octlet**
Within an octlet, the most significant quadlet is that which is transmitted first and the least significant quadlet is that which is transmitted last on the Serial Bus, as shown below.

```
| most significant quadlet |  |  |
```

```
| least significant quadlet |
```

**Figure 3** Quadlet ordering
4. Direct Print Model

The model described in this document has a layered structure as shown in Figure 4. The Application uses Thin Session service for transaction layer connection, and uses Direct Print Command Set for printing. To ensure that an image is printed, this document specifies the data transfer protocol and printing command set. By using the protocol and the command set, any image source devices can print their images to any image output devices.

![Figure 4 Direct Print Model](image)

- **Direct Print Command Set**: The definition of commands used by Direct Print Application.
- **Thin Session**: Responsible for management of a connection and segmentation/reassembly of Application data to optimize the resources used by the Thin Transaction.
- **Thin Transaction**: Responsible for transmission/receipt of data issued by the Thin Session.
Direct Print Protocol (DPP) has the following features:

1) DPP is a layered model which has the Direct Print Command Set for printing and the Thin Protocol for data transfer.

2) Direct Print Command Set defines mandatory and optional functions for printing.

3) Direct Print Command Set supports default values for printing parameters, which enables printing without inquiry of capability.

4) Direct Print Command Set includes commands to support inquiry of target capability, thus achieving print output according to the capabilities of various printing devices.

5) Thin Protocol has a symmetrical structure and each node can send and receive Application data.

6) Thin Protocol dynamically allocates CSR register space when establishing a connection.

7) Thin Protocol is capable of selecting negotiation parameters at the time of establishing a connection.

Direct Print Protocol can achieve a print application on IEEE 1394 layer when being used as an entire set. However, the specifications of protocols, Direct Print Command Set and Thin Protocol, are designed to be independent of each other, so that they can be individually used.
5. Thin Protocol

5.1. Overview

Thin Protocol is a symmetric transfer protocol on IEEE 1394 layer. Both the Initiator node (which issues a connect request) and the Responder node (which receives a connect request) can send and receive data. Thin Protocol has been designed to achieve the following abilities:

- Symmetric communications.
- Dynamic allocation of resources used for data transfer (register space, information about connection) when establishing a connection. This enables the optimal utilization of resources in accordance with the Application.
- Support for segmentation and reassembly of Application data for transfer.
- Use on any other network protocol.
- Multiple independent data transfer over one connection.

5.2. Service Model

Thin Protocol is a multiple layer model. Each layer provides services to communicate between layers. Four types of service are defined by this document.

- **Request service**: Request service is a communication from the higher layer to the lower layer to request the lower layer to perform some action. In this document, request service is abbreviated to ‘req’.
- **Indication service**: Indication service is a communication from the lower layer to the upper layer to indicate a change of state or other event detected by the lower layer. In this document, indication service is abbreviated to ‘ind’.
- **Response service**: Response service is a communication from the higher layer to the lower layer to respond to an indication. In this document, response service is abbreviated to ‘rsp’.
- **Confirmation service**: Confirmation service is a communication from the lower layer to the upper layer to confirm request service. In this document, confirmation service is abbreviated to ‘cnf’.
If all four service types exist, the service model that occurred within the Requester node and the Executor node are related as shown by the following figure.

![Service model](image)

**Figure 5 Service model**

### 5.3. Thin Protocol Layers

Thin Protocol has two layers; Thin Session and Thin Transaction. The functions of each layer are described below.

#### 5.3.1. Thin Session Layer

Thin Session layer realizes these functions.
- Segmentation and reassembly of Application data to optimize the resources used by the Thin Transaction.

#### 5.3.2. Thin Transaction Layer

Thin Transaction layer realizes these functions.
- Transmission and receipt of data issued by the Thin Session.
- Management of the resources that is used for transfer.

#### 5.3.3. Thin Session Service

Thin Session provides the services listed below to the Application. Thin Session services are described with ‘S_’ prefix, like S_Connect.req, S_Disconnect.ind, and so on.
- Connection establishment < S_Connect (req, ind, rsp, cnf) >
- Disconnection < S_Disconnect (req, ind) >
- Execution of Application command < S_Command (req, ind, rsp, cnf) >
- Abort execution of Application command < S_Abort (req, ind) >
- Indication of CommandID for identifying Application commands < S_CommandID (ind) >
- Indications of errors < S_Error (ind) >
5.3.4. Thin Transaction Service

Thin Transaction provides the services listed below to Thin Session. Thin Transaction services are described with ‘T_’ prefix, like T_Connect.req, T_Disconnect.ind, and so on.

- Connection establishment and allocation of resources < T_Connect (req, ind, rsp, cnf) >
- Re-establishment of a connection < T_Reconnect (req, ind, rsp, cnf) >
- Disconnection and release of resources < T_Disconnect (req, ind) >
- Transfer of data < T_SDUSend (req, ind, rsp, cnf) >
- Abort data transfer < T_SDUStop (req, ind, rsp, cnf) >
- Indication of errors < T_Error (ind) >
5.4. Thin Protocol Sequence

The flow of procedure using Thin Protocol is as follows. Each process will be described in section 5.5 and later in this document.

1) Connect

This step is provided to connect two nodes and allocate resources. What is meant by “resources” here is the register space for transmission/receipt of data and information about the connection. Information for the allocation of resources is determined by negotiation parameters at the time of establishing a connection.

   Sequence of connection
   i) The node that starts a connection issues a connect request.
   ii) The node that has received a connect request issues a connect response.

2) Command

While a connection is established, one node may request the other node to execute a command and may transmit data.

   Sequence of Command issue
   i) The node that wants to send a command issues a command request.
   ii) The node that has received a command request executes the command and issues a command response.

3) Disconnect

This step is provided to terminate a connection. At this process, the resources allocated at the time of establishing a connection are released.

   Sequence of Disconnect
   i) The node that terminates a connection issues a disconnect request.

The procedure described above can be illustrated as a diagram of sequence shown in Figure 6. The node which starts a connection, the node which issues a command and the node which terminates a connection may be either of the two nodes, though described as the same node in the diagram.
Figure 6  Thin protocol sequence
5.5. Thin Protocol Segmentation

Thin Protocol has the capability of segmentation and reassembly for both Thin Transaction and Thin Session. Each type of segmentation is used for the following objectives.

**Thin Session segmentation:** Segments Application data into multiple pieces of Segment data in accordance with the size of the SDU register that is allocated at the time of establishing a connection (described in later sections.)

**Thin Transaction segmentation:** Segments the data requested by the Thin Session (SDU: Segment Data Unit) so as to have the maximum transfer data size of the IEEE1394 write transaction.

---

**Figure 7** Thin Protocol segmentation
5.6. Register Space
Thin Protocol uses the following three registers. Thin protocol compliant devices shall implement these registers. See section 5.8.1.1 for the procedure of allocating register spaces.

- Connection register
- SDU management register
- SDU register

![Diagram of register space]

5.6.1. Connection Register
The Connection register shall be used when establishing a connection. This register space size is fixed (512 bytes). The base address of this register space can be determined by referring to Configuration ROM (see section 7.1.4.6.)

Thin Protocol packet using the Connection register
- Connect Req: a connect request
- Connect Rsp: a connect response
- Reconnect Req: a reconnect request
- Reconnect Rsp: a reconnect response
- Disconnect Req: a disconnect request

In the following sequence diagrams, these Thin Protocol packet transfer using the Connection register will be marked with a double-line ( ).
5.6.2. SDU Management Register

The SDU management register shall be used for controlling the transfer of an SDU (Segment Data Unit: a segment of Application data with a Thin Session header). The SDU management register has two parts, SDU control space and SDU response space. SDU control space is used for storing completion notification of an SDU transfer and abort of an SDU transfer. SDU response space is used for storing the response of an SDU transfer and the rejection of an SDU transfer.

This register space size is fixed (4 bytes for each register space, so the SDU management register size is 8 bytes). The base address of this register space is dynamically determined by the procedure described in section 5.8.1.1.

**Thin Protocol packet using the SDU management register**

**SDU control space**
- SDU Comp: completion of SDU transfer.
- SDU Stop: abort an SDU transfer.

**SDU response space**
- SDU Ready: response of ready to write to the SDU register.
- SDU Reject: rejection of SDU transfer.

In the following sequence diagrams, the Thin Protocol packet transfer using the SDU management register will be marked with a dotted line (            ).

5.6.3. SDU Register

The SDU register shall be used for transferring an SDU. The size and base address of this register space is dynamically allocated by the procedure described in section 5.8.1.1.

**Thin Protocol packet using the SDU register**

**SDU Send**: SDU Transfer

In the following sequence diagrams, the Thin Protocol packet transfer using the SDU register will be marked with a bold line (                    ).
5.6.4. Register Space Summary

Table 1 Register space summary

<table>
<thead>
<tr>
<th>Register</th>
<th>Thin Protocol packet</th>
<th>Arrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection register</td>
<td>Connect Req / Rsp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reconnect Req / Rsp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Disconnect Req</td>
<td></td>
</tr>
<tr>
<td>SDU management register</td>
<td>SDU Ready, SDU Reject (SDU response space)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDU Comp, SDU Stop (SDU control space)</td>
<td></td>
</tr>
<tr>
<td>SDU register</td>
<td>SDU Send</td>
<td></td>
</tr>
</tbody>
</table>

5.7. Connection Information

The Thin Protocol shall retain information about a connection, such as node_id, Machine ID, Vendor ID, Command Set ID, Transfer Mode, etc., while the connection is established. Node_id can be determined by analyzing the 1394 transaction header. Machine ID is provided by the Application, and serves as the ID of the node. In this document, node_unique_id is used as the Machine ID. Machine ID can be determined by analyzing the Thin Protocol Packet. Other information is exchanged as negotiation information in the connect sequence (described in 5.11.3). If a bus reset occurs, connection information is updated in a reconnect sequence. After that, communication will be resumed using the new node_id (described in 5.8.2).

Table 2 Connection information

<table>
<thead>
<tr>
<th>node_id</th>
<th>Machine ID</th>
<th>Vendor ID</th>
<th>Command set</th>
<th>Transfer mode</th>
<th>.....</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>xxxx xxxx</td>
<td>xx xxxx</td>
<td>xx</td>
<td>1394,Async,PUSH</td>
<td>.....</td>
</tr>
</tbody>
</table>
5.8. **Connection Sequence**

This section describes the sequence of the Thin Protocol connection, which consists of establishing, resuming and terminating a connection between nodes.

5.8.1. **Connect**

The Thin Protocol shall use this sequence to establish a connection. In a connect sequence, each node shall allocate necessary resources according to the negotiation information (described in section 5.11.3) contained in a connect request packet and a connect response packet. What is meant by resources for the Thin Protocol is a register space allocation used as the SDU management register and the SDU register (described in sections 5.6.2 and 5.6.3), buffer space, and information about the connection (described in section 5.7). The algorithm of allocating register space is described in section 5.8.1.1 and later in this document.

The connect sequence shall use the following steps.

1) The Application of the node which wants to make a connection (Initiator node) issues a connect request (S_Connect.req).

2) The Thin Session of the Initiator node issues a connect request (T_Connect.req) to the Thin Transaction.

3) The Thin Transaction of the Initiator node transfers a connect request packet (Connect Req) that includes negotiation information.¹

4) The Thin Transaction of the node which has received a connect request packet (Responder node) issues a connect indication (T_Connect.ind) to the Thin Session.

5) The Thin Session of the Responder node issues a connect indication (S_Connect.ind) to the Application.

6) The Application of the Responder node checks whether a communication is permitted or not and issues a connect response (S_Connect.rsp) to the Thin Session.

7) The Thin Session of the Responder node issues a connect response (T_Connect.rsp) to the Thin Transaction.

8) The Thin Transaction of the Responder node transfers a connect response packet (Connect Rsp) after allocating necessary resources.²

---

¹ A connect request packet and a connect response packet should be written to the connection register by using one write transaction. Therefore, the devices supporting the Thin Protocol shall receive block write transaction.

² If the connection is rejected by the higher layer, the Thin Transaction shall not allocate resources.
9) The Thin Transaction of the Initiator node, at the time of receiving the connect response packet, issues a connect confirmation (T_Connect.cnf) to the Thin Session after allocating necessary resources.³

10) The Thin Session issues a connect confirmation (S_Connect.cnf) to the Application⁴.

---

³ If a connect response is nack, the Thin Transaction of the Initiator node shall not allocate resources and shall issue a connect confirmation of connect reject to the Thin Session.

⁴ When the Thin Session does not receive a connect confirmation from the Thin Transaction within a connect timeout period (5 seconds), it shall issue an error indication to Application instead of a connect confirmation (this sequence is detailed in section 5.10.1.)
5.8.1.1. **Dynamic Register Allocation Algorithm**
This section describes the procedure for allocating a register space. In the description below, the node which issues a connect request is referred to as the “Initiator node” and the node which issues a connect response is referred to as the “Responder node”.

5.8.1.1.1. **Dynamic Register Allocation Sequence**
The following steps describe the dynamic register allocation sequence.

1) The Initiator node sends a connect request:
   i) Transfers a connect request packet (Connect Req) including the negotiation information of the following parameters:
      - SDU management register address of Initiator node (Addr A).
      - SDU register address of Initiator node (Addr B1).
      - SDU register size of Initiator node (Size B1). This size is determined by the maximum contiguous address space that the Initiator node can allocate for segment data receiving, and shall be at least 512 bytes.
      - SDU register size of Responder node (Size D1). This size is determined by the maximum segment data size that the Initiator node wishes to send, and shall be at least 512 bytes.

![Figure 10: Dynamic register allocation sequence - 1](image-url)
2) Responder node sends a connect response:
   i) Allocates the SDU register and SDU management register. If the Responder node can allocate
      the size requested in the connect request packet (Size D1), the Responder node shall allocate
      the SDU register with the requested size. If not, Responder node shall allocate the SDU
      register with the maximum size that Responder node can allocate (shall be at least 512 byte).
   ii) Determines the SDU register size of the Initiator node. The size of the SDU register shall be
       equal to the size requested in a connect request packet (Size B1) or the maximum segment data
       size that the Responder node wishes to send, whichever is smaller. (It shall be at least 512
       bytes).
   iii) Determines the maximum segment data transfer size of the Responder node from the size
       determined in step ii).
   iv) Transfers a connect response packet (Connect Rsp) including the negotiation information of
       the following parameters:
       - SDU management register address of Responder node (Addr C).
       - SDU register address of Responder node (Addr D2).
       - SDU register size of Responder node (Size D2). This size is the register size allocated
         in step i).
       - SDU register size of Initiator node (Size B2). This size is the register size determined
         in step ii).

Figure 11 Dynamic register allocation sequence - 2
3) Initiator node receives a connect response:
   i) Allocates the SDU register and SDU management register. The size of the SDU register shall be equal to the size contained in a connect response packet (Size D2).
   ii) Determines the maximum segment data transfer size of the Responder node from the register size contained in a connect response packet (Size B2).

As result of this sequence, these registers shall be allocated at the following CSR space.

- **Initiator node SDU management register**
  - base address: Addr A, size: 8 bytes (fixed)
- **Initiator node SDU register**
  - base address: Addr B2, size: Size B2
- **Responder node SDU management register**
  - base address: Addr C, size: 8 bytes (fixed)
- **Responder node SDU register**
  - base address: Addr D2, size: Size D2

**Figure 12** Dynamic register allocation sequence - 3
5.8.2. Reconnect

5.8.2.1. Reconnect sequence

When a 1394 connection is broken by a bus reset or unexpected error, the Thin Protocol may attempt to resume a connection by using this sequence. In a reconnect sequence, the Machine ID (node_unique_id) of a disconnected node shall be sent to another node to re-establish the connection.

When the 1394 connection is broken, a reconnect timer shall be started. The node that wants to reconnect shall issue a reconnect request immediately. Unless a reconnection is performed within a reconnect timeout period (5 seconds) after notification of the error, the connection shall be discarded and resources shall be released.

The node that has received a reconnect request shall return a reconnect ack if it retains the condition that existed just before the bus reset. If it has already discarded the condition, the node shall return a reconnect nack.

After completing a reconnect sequence, the Thin Session resumes the data transfer. If the Thin Session is sending a segment of data, the transfer shall be started from the top of the SDU that had its transfer interrupted during the reconnect sequence.

The reconnect sequence shall use the following steps. The SDU transfer procedure, shown in the sequence diagram (Figure 13), is described in section 5.9 and later.

1) When a bus reset occurs, the Thin Transaction of each node indicates an error (T_Error.ind) to the Thin Session.

2) After receiving indication of the error, the node that wants to reconnect (Sender node) seeks the new node_id of the previously connected node. And then the Thin Session of the Sender node issues a reconnect request (T_Reconnect.req) to the Thin Transaction within the reconnect timeout period.

3) The Thin Transaction of the Sender node transfers a reconnect request packet (Reconnect Req).

5 A reconnect timer shall be restarted if a second bus reset occurs before a reconnection is established.

6 If the previously connected node does not exist in the bus, the Thin Session of the Sender node shall discard all resources of the connection and issue an error indication of reconnect reject to Application.

7 A reconnect request packet and a reconnect response packet should be written to the connection register by using one write transaction.
4) The Thin Transaction of the node that has received a reconnect request packet (Receiver node) issues a reconnect indication (T_Reconnect.ind) to the Thin Session.\(^8\)

5) The Thin Session of the Receiver node determines if the reconnection is permitted or not (i.e. whether the previous resources are available or not). If the reconnection is permitted, the node issues a reconnect response (T_Reconnect.rsp) containing a reconnect ack to the Thin Transaction. If not, it issues a reconnect response containing a reconnect nack.

6) The Thin Transaction of the Receiver node transfers a reconnect response packet (Reconnect Rsp).

7) The Thin Transaction of the Sender node receives a reconnect response packet. If the reconnection is permitted, it issues a reconnect confirmation (T_Reconnect.cnf) to the Thin Session\(^9\).

8) The Thin Session of the Sender node resumes data transfer from the top of the SDU that had its transfer interrupted during the reconnect sequence\(^{10}\).

---

\(^8\) If a reconnect request packet is received after issuing a reconnect request packet or a reconnect response packet, the Thin Transaction shall return a reconnect ack. If a reconnect request packet is received after releasing resources, the Thin Transaction shall return a reconnect nack.

\(^9\) If a reconnect response is nack, the Thin Transaction of the Sender node shall release the resources and issue a reconnect confirmation of reconnect reject to the Thin Session.

\(^{10}\) When the Thin Session does not receive a reconnect confirmation from the Thin Transaction within a reconnect timeout period (5 seconds), it shall issue an error indication to the Application (this sequence is detailed in section 5.10.2.)
Figure 13  Reconnect sequence
5.8.3. Disconnect

The Thin Protocol uses this sequence to terminate a connection. In a disconnect sequence, the resources allocated at the time of establishing a connection shall be released.

The disconnect sequence shall use the following steps.

Note that this sequence shall not expect a response from the Receiver node.

1) The Application of the node that performs the disconnect (Sender node) issues a disconnect request (S_Disconnect.req).

2) The Thin Session of the Sender node issues a disconnect request (T_Disconnect.req) to the Thin Transaction.\(^\text{11}\)

3) The Thin Transaction of the Sender node releases resources. At this time, any write transaction to the SDU register shall be ignored.\(^\text{12}\) The Thin Transaction of Sender node, then transfers a disconnect request packet (Disconnect Req).\(^\text{13}\)

4) The Thin Transaction of the node that has received a disconnect request packet (Receiver node) releases resources and then issues a disconnect indication (T_Disconnect.ind) to the Thin Session.\(^\text{14}\)

5) The Thin Session of the Receiver node issues a disconnect indication (S_Disconnect.ind) to the Application.

\(^{11}\) If the SDU transfer is not completed, the Thin Session shall stop the transfer by using the abort sequence (see section 5.9.5) before issuing a disconnect request.

\(^{12}\) If the Thin Transaction uses an interruption for receiving a write transaction, the interruption shall be ignored.

\(^{13}\) A disconnect request packet should be written to the connection register by using one write transaction.

\(^{14}\) If the connection requested to disconnect is not established yet or already terminated, the disconnect request shall be ignored.
Figure 14 Disconnect sequence

1) S_Disconnect.req  →  2) T_Disconnect.req
                  Resource Release

3) Disconnect Req  →  4) T_Disconnect.ind
                  Resource Release

5) S_Disconnect.ind → Disconnect

Sender node
Thin Session Thin Transaction

Receiver node
Thin Transaction Thin Session
5.9. Asynchronous Transfer

The Thin Protocol has the capability of segmentation and reassembly for both the Thin Transaction and Thin Session. Each type of segmentation shall be used for the objectives below.

**Thin Session segmentation:** Segments Application data into multiple pieces of Segment data in accordance with the size of the SDU register.

The Thin Session determines if Application data segmentation is needed or not. When the Application data size is small enough to be transferred all at once using the SDU register, the Thin Session shall use a single data transfer sequence (described in section 5.9.1.1). When the Application data size is larger than the size of the SDU register, the Thin Session shall use the segment data transfer sequence (described in section 5.9.1.2).

**Thin Transaction segmentation:** Segments the data requested by the Thin Session (SDU; Segment Data Unit), taking advantage of the maximum transfer data size of the IEEE1394 write transaction. The data transfer sequence of the Thin Transaction is described in sections 5.9.1.3 and later.

![Figure 15 Thin Protocol segmentation](image)
5.9.1. Thin Protocol Data Transfer

5.9.1.1. Thin Session Single Data Transfer Sequence

If the Application data size is small enough to be transferred all at once using the SDU register, the Thin Session shall issue an SDU transfer request (T_SDUSend.req) only once to the Thin Transaction.

The SDU transfer procedure of the Thin Transaction is detailed in section 5.9.1.3. In the sequence diagram (Figure 16), the SDU transfer is marked with a bold arrow ( ).

The single data transfer sequence shall use the following steps.

1) The Application of the node that transfers data (Sender node) issues a Thin Session service with Application data.

2) The Thin Session of the Sender node issues an SDU transfer request (T_SDUSend.req) to the Thin Transaction.

3) The Thin Transaction of the Sender node transfers an SDU (see section 5.9.1.3.).

4) The Thin Transaction of the node that has received an SDU (Receiver node) issues an SDU transfer indication (T_SDUSend.ind) to the Thin Session.

5) The Thin Session of the Receiver node issues a Thin Session service with the received Application data if the SDU is accepted.

6) The Thin Session of the Receiver node issues an SDU transfer response (T_SDUSend.rsp) describing an SDU ready state, when it becomes possible to write to the SDU register. If the Thin Session cannot accept the SDU, the Thin Session issues an SDU transfer response describing an SDU reject.

7) The Thin Transaction of the Receiver node transfers an SDU transfer ready packet (SDU Ready) if the response from the Thin Session describes an SDU ready state. If not, it transfers an SDU transfer reject packet (SDU Reject).

8) The Thin Transaction of the Sender node issues an SDU transfer confirmation (T_SDUSend.cnf) to the Thin Session. The Thin Session of the Sender node may then send another SDU if necessary. 15

---

15 If this confirmation is SDU reject, the Thin Session may retry to send the same data. If the Thin Session does not retry, it shall issue an error indication to the Application. When an error is indicated, the Application shall issue an abort request (see section 5.9.5).
Figure 16 Thin Session Single data transfer
5.9.1.2. Thin Session Segment Data Transfer Sequence

If the Application data cannot be transferred all at once using the SDU register, the Thin Session shall segment the Application data into segments that are equal to or less than the size of the SDU register. If the case that Application data is segmented, the Thin Session shall issue an SDU transfer request (T_SDUSend.req) to the Thin Transaction for each data segment. The data transfer procedure of the Thin Transaction is detailed in section 5.9.1.3. In the sequence diagram (Figure 17), the SDU Transfer is marked with a bold arrow ( ).

The segment data transfer sequence shall use the following steps.

1) The Application of the node that transfers data (Sender node) issues a Thin Session service with Application data.

2) The Thin Session of the Sender node segments the Application data into multiple pieces of Segment data, each having a transferable data size by using the SDU register.

3) The Thin Session of the Sender node issues an SDU transfer request (T_SDUSend.req) to the Thin Transaction for each piece of Segment data.

4) The Thin Transaction of the Sender node transfers an SDU (see section 5.9.1.3.).

5) The Thin Transaction of the node that has received an SDU (Receiver node) issues an SDU transfer indication (T_SDUSend.ind) to the Thin Session.

6) The Thin Session of the Receiver node issues an SDU transfer response (T_SDUSend.rsp) describing an SDU ready state, when it becomes possible to write to the SDU register. If the Thin Session cannot accept the SDU, the Thin Session issues an SDU transfer response describing an SDU reject.

7) The Thin Transaction of the Receiver node transfers an SDU transfer ready packet (SDU Ready) if the response from the Thin Session is describing an SDU ready state. If not, it transfers an SDU transfer reject packet (SDU Reject).

8) The Thin Transaction of the Sender node issues an SDU transfer confirmation (T_SDUSend.cnf) to the Thin Session.

9) When the confirmation from the Thin Transaction is describing an SDU ready state, the Thin Session of the Sender node sends the next SDU up to the last SDU packet of Application data in order by using the same sequence of steps 3) to 8) 16.

16 If this confirmation is an SDU reject, the Thin Session may retry to send the same SDU. If the Thin Session does not retry, it shall issue an error indication to the Application. When an error is indicated, the Application shall issue an abort request (see section 5.9.5).
10) The Thin Session of the Receiver node reassembles the Application data and issues indications of received Application data, when all the pieces of SDU have been received and all SDU transfers are accepted\(^{17}\).

---

\(^{17}\) Each SDU contains a field of information about its location in the segmented data (first/interim/last). This allows the Thin Session of the Receiver node to recognize whether all the SDU packets have arrived or not. Details of the Thin Session format is described in section 5.11.7.
5.9.1.3. Thin Transaction Data Transfer Sequence

The Thin Transaction shall segment the data requested by the Thin Session into multiple pieces of data, taking advantage of the maximum transfer data size of the IEEE 1394 Transaction (max_rec). The data transfer sequence of the Thin Transaction shall use the following steps.

1) The Thin Session of the node that transfers data (Sender node) issues an SDU transfer request (T_SDUSend.req).
2) The Thin Transaction of the Sender node segments the data into an SDU, using the maximum data transfer size of a write transaction (according to max_rec field of Configuration ROM described in section 7.1.2)\(^\text{18}\), and issues multiple write transactions.\(^\text{19}\)
3) When the transfer of the SDU has been completed, the Thin transaction of the Sender node transfers an SDU transfer completion packet (SDU Comp).
4) The Thin Transaction of the node that has received the data (Receiver node) issues an SDU transfer indication (T_SDUSend.ind) to the Thin Session.
5) The Thin Session of the Receiver node issues an SDU transfer response (T_SDUSend.rsp) describing an SDU ready state, when it becomes possible to write to the SDU register. If the Thin Session cannot accept the SDU, the Thin Session issues an SDU transfer response describing an SDU reject.
6) The Thin Transaction of the Receiver node transfers an SDU transfer ready packet (SDU Ready) if the response from the Thin Session is describing an SDU ready state. If not, it transfers an SDU transfer reject packet (SDU Reject).
7) The Thin Transaction of the Sender node issues an SDU transfer confirmation (T_SDUSend.cnf) to the Thin Session. The Thin Session of the Sender node may then send another SDU if necessary.\(^\text{20}\)

\(^\text{18}\) Write transaction sizes smaller than max_rec should be allowed, but not preferred.
\(^\text{19}\) Two or more write transactions in the same address should be allowed. The receiver shall treat the write transactions in any order. The sender may send write transactions in any order but it is not preferred.
\(^\text{20}\) If the Receiver node does not want the Sender node to write to the SDU register, it shall not send SDU Ready. However, if the Receiver node does not send SDU Ready within a data transfer timeout period, a timeout error shall occur in the Sender node (the timeout sequence is described in section 5.10).
Figure 18    Thin Transaction data transfer

In this document, SDU Send and SDU Comp of this data transfer procedure (Steps 2 and 3) is described as "SDU Transfer", and marked with a bold arrow ( → ) in the sequence diagrams.
5.9.1.4. Thin Transaction SDU Transfer Stop Sequence

The Thin Session shall issue an SDU transfer stop to terminate the SDU transfer.

The SDU transfer stop sequence shall use the following steps.

1) The Thin Session of the node that wants to terminate an SDU transfer (Sender node) issues an SDU transfer stop request (T_SDUStop.req).
2) The Thin Transaction of the Sender node sends an SDU transfer stop packet (SDU Stop).
3) The Thin Transaction of the node that has received the SDU transfer stop packet (Receiver node) issues an SDU transfer stop indication (T_SDUStop.ind) to the Thin Session.
4) The Thin Session of the Receiver node clears the receiving SDU data, and then issues an SDU transfer stop response (T_SDUStop.rsp) to the Thin Transaction.
5) The Thin Transaction of the Receiver node sends an SDU transfer ready packet (SDU Ready).
6) The Thin Transaction of the Sender node issues an SDU transfer stop confirmation (T_SDUStop.cnf) to the Thin Session when it has received an SDU transfer ready packet. The Thin Session of the Sender node may then send another SDU if necessary.

![Diagram of SDU transfer stop sequence]

**Figure 19** Thin Transaction data transfer stop
5.9.2. Application Command Transfer

For command execution, two transfer sequences are used.

- The node that wants to request command execution (Requester node) issues a command request.
- The node that receives a command request (Executor node) executes the command, and issues a command response with the result of the command execution.

The command request and command response are each treated as Application data. The Thin Session and Thin Transaction perform the transfer operation as described in sections 5.9.1.1 through 5.9.1.3.

It is possible to define a command request that does not expect a command response.

When the Application issues a command request, the Thin session shall apply a sequential ID number (Command ID) for identifying each command. When the Application issues a command response, the Application shall assign this same Command ID number as the corresponding command request.

One example of the Application command transfer sequence is described below. The procedure is exemplified by the case in which the Application data size of a command request is so large that it shall be segmented by the Thin Session, and the Application data size of command response is small enough to be transferred without segmentation by the Thin Session.

1) The Application of the Requester node issues a command request (S_Command.req) to the Thin Session.

2) The Thin Session of the Requester node applies a sequential ID number (Command ID) for identifying each command, and issues an indication of an assigned Command ID (S_CommandID.ind) to the Application.

3) The Thin Session and Thin Transaction of the Requester node perform the data transfer by using the sequence described in sections 5.9.1.2 and 5.9.1.3, respectively.

4) The Thin Session of the Executor node issues a command indication (S_Command.ind) to the Application when the data of the Application command has arrived.

5) The Application of the Executor node executes a received command, and issues the results of the command execution as a command response (S_Command.rsp).

6) The Thin Session and Thin Transaction of the Executor node perform the data transfer by using the sequence described in sections 5.9.1.1 and 5.9.1.3, respectively.

7) The Thin Session of the Requester node issues a command confirmation (S_Command.cnf) to the Application when the data on the results of the command execution has arrived.
Figure 20  Application command transfer
5.9.3. Multiple Data Transfer

The Thin Protocol uses Command IDs to distinguish one application command from another. Therefore, the Application may issue multiple commands simultaneously. The number of commands that can be issued is specified by the command set (e.g. the DPP command set specifies 2 commands can be issued).

The order of each command request is unspecified.

When the node receives a command request while receiving other command requests, the Thin Session of the node issues the command indication to the Application if the node has the capability of handling the command request. If the node does not have the capability, the Thin Session of the node shall issue an SDU transfer response describing an SDU reject.

When the node accepts multiple commands, the order of the command responses of each command is unspecified. (This means that the command confirmation of the second command may be processed before the command confirmation of the first command.)

The sequence diagram below shows the case where the Application issues multiple commands.

![Sequence Diagram](Figure 21 Multiple data transfer)
5.9.4. Bi-directional Data Transfer

The Thin Protocol ensures bi-directional simultaneous transfer of data between two nodes by using an SDU transfer completion (SDUComp) and an SDU transfer ready (SDUReady). Therefore, an Application may send one command while receiving another command.

When the node receives a command request while sending another command request, the Thin Session of the node issues a command indication to the Application if the node has the capability of handling the command request. If the node does not have the capability, the Thin Session of the node shall issue an SDU transfer response describing an SDU reject.

The order of a sending command request and a command response of a receiving command request is unspecified. (This means that the command confirmation may be sent after completing the command request transfer.)

The sequence diagram below shows the sequence for issuing a command in a bi-directional manner.

---

**Figure 22** Bi-directional data transfer
5.9.5. Abort

The Application may abort command execution at any time (e.g. by using Cancel command of DPP Command set\(^{21}\)). Both the node sending a command request and the node receiving a command request may send this command abort request. The Application shall issue a command abort request with the Command ID of the Application command that is requested to be aborted. The Thin Session may issue an SDU transfer stop request if the Thin Transaction is sending an SDU containing Application data for a command that has been requested to be aborted.

The abort sequence shall use the following steps.

Note that this sequence shall not expect a response from the Receiver node.

1) The Application of the node that wants to abort the command execution (Sender node) issues a command abort request (S_Abort.req) to the Thin Session.

2) If the Thin Transaction of the Sender node is sending an SDU (e.g. before receiving SDU Ready) that has the same Command ID as the command abort request, the Thin Session may issue an SDU transfer stop request (this sequence is described in section 5.9.1.4). Otherwise, the Thin Session shall wait to receive SDU Ready (or SDU Reject) before sending an SDU transfer request.

3) The Thin Session of the Sender node issues an SDU transfer request (T_SDUSend.req) to the Thin Transaction for transferring a higher layer abort command (This format is described in section 5.11.8.1) by using the sequence described in sections 5.9.1.2 and 5.9.1.3, respectively.

4) If the Thin Transaction of the node that receives a higher layer abort command (Receiver node) is sending an SDU (e.g. before receiving SDU Ready) that has the same Command ID as the command abort request, the Thin Session may issue an SDU transfer stop request (this sequence is described in section 5.9.1.4). Otherwise, the Thin Session shall wait to receive SDU Ready (or SDU Reject) before issuing a command abort indication.

5) The Thin Session of the Receiver node issues a command abort indication (S_Abort.ind) to the Application.

\(^{21}\) Note that the Cancel command of DPP Command set shall be issued by using the abort sequence.
Figure 23  Abort sequence

1) S_Abort.req

2) This sequence is used only when the Thin Session wants to stop SDU transfer.

3) T_SDUSend.req
(abort command)

4) This sequence is used only when the Thin Session wants to stop SDU transfer.

5) S_Abort.ind
5.10. Timeout Sequence

The Thin Protocol uses several timeout periods for solving abnormal conditions. The Thin protocol compliant devices shall implement these timeouts. These timeouts shall be realized in the Thin Session protocol.

5.10.1. Connect Timeout

Timeout value: 5 seconds.

The connect timeout occurs when the Thin Session issues a connect request and a connect response service from the Thin Transaction is not issued within the specified period. If this timeout occurs, the Thin Session shall issue an error indication to the Application, and shall issue a disconnect request to the Thin Transaction (for releasing the resources of the receiver node). This disconnect request causes a disconnect sequence described in section 5.8.3.

![Connect timeout diagram](image-url)
5.10.2. Reconnect Timeout

Timeout value: 5 seconds

The Thin timeout occurs when a reconnect sequence is not completed within the specified period after bus reset. If this timeout occurs, the Thin Session shall release the resources, issue an error indication to the Application, and issue a disconnect request to the Thin Transaction (for releasing the resources of the receiver node). This disconnect request causes a disconnect sequence described in section 5.8.3.

![Reconnect Timeout Diagram](image)

**Figure 25** Reconnect timeout
5.10.3. **Idle Timeout**

Timeout value: determined by negotiation when establishing a connection (default: 30 min).

This timeout occurs when the Thin Protocol connection has been established and no service from/to the Thin Transaction has been issued within the specified period. This timeout shall be restarted when any services has been issued from/to the Thin Transaction. If this timeout occurs, the Thin Session shall issue an error indication to the Application. The Application may issue a disconnect request if an error indication is issued. The sequence is specified by the Application.

![Diagram of Idle Timeout](image)

**Figure 26**  Idle timeout
5.10.4. Data Transfer Timeout

Timeout value: determined by negotiation when establishing a connection (default: 10min).

This timeout occurs when, during the process of data transfer, no service from/to the Thin Transaction has been issued within the specified period. Specifically, this timeout occurs in the following situations:

- The Thin Session issues an SDU transfer request, but an SDU transfer confirmation is not issued by the Thin Transaction.
- The Thin Session receives a segment data and replies with an SDU transfer response, but an SDU transfer indication of the next segment data is not issued.

If this timeout occurs, the Thin Session shall issue an error indication to the Application. The Application may issue a disconnect request if an error indication is issued. The sequence is specified by the Application.
Figure 27  Data transfer timeout - 1

Figure 28  Data transfer timeout - 2
Note that this timeout shall be restarted when any services has been issued from/to the Thin Transaction. So, when a segment data is received while another SDU is being sent, the timeout shall be restarted at the time that the last service is received.
5.11. Thin Protocol Format
The Thin Protocol Format is described in this section.
In the Thin Protocol, the Thin Session attaches the Thin Session field to the Application data. The Thin Transaction transfers the packet with a 1394 write transaction.
The Thin Session field format and Thin Transaction transfer operation are described in the following sections.

5.11.1. Thin Session Service
The Thin Session provides the following services to the Application:
  * Connect establishment < S_Connect (req, ind, rsp, cnf) >
  * Disconnect < S_Disconnect (req, ind) >
  * Execution of Application command < S_Command (req, ind, rsp, cnf) >
  * Abort execution of Application command < S_Abort (req, ind) >
  * Indication of CommandID for identifying Application commands < S_CommandID (ind) >
  * Indication of errors < S_Error (ind) >

The Thin Session field format for realizing the services above is described below.
The services of indication of CommandID and indication of error have no packet, since they are an internal process issued by the Thin Session to the Application.

5.11.2. Thin Session Field Format
The Thin Session field is composed of a Session ID, MsgType and one or more information tag(s). The Session ID is used for distinguishing the session. In this document, Session ID shall be set to 00h.
MsgType is used for representing the kind of message.
Information tag is used for describing various information, and is composed of InfType for discerning the information, and Information value.

<table>
<thead>
<tr>
<th>Session ID</th>
<th>MsgType</th>
<th>InfType</th>
<th>Information</th>
<th>InfType</th>
</tr>
</thead>
</table>

**Figure 30** Thin Session field format
The MsgType and InfType are defined below. All values other than the ones mentioned below are reserved.

### Table 3  MsgType defined value

<table>
<thead>
<tr>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10h</td>
<td>Connect request</td>
</tr>
<tr>
<td>11h</td>
<td>Connect confirmation</td>
</tr>
<tr>
<td>12h</td>
<td>Connect rejection</td>
</tr>
<tr>
<td>20h</td>
<td>Data</td>
</tr>
<tr>
<td>21h</td>
<td>Data abort</td>
</tr>
<tr>
<td>30h</td>
<td>Disconnect request</td>
</tr>
<tr>
<td>40h</td>
<td>Reconnect request</td>
</tr>
<tr>
<td>41h</td>
<td>Reconnect confirmation</td>
</tr>
<tr>
<td>42h</td>
<td>Reconnect rejection</td>
</tr>
</tbody>
</table>

### Table 4  InfType defined value

<table>
<thead>
<tr>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Version</td>
</tr>
<tr>
<td>40h</td>
<td>Connection data</td>
</tr>
<tr>
<td>80h</td>
<td>Application data</td>
</tr>
</tbody>
</table>
5.11.3. Negotiation Information

There are several types of information, such as timeout values, register address and size exchanged during the connect sequence (Table 5). All devices compliant with the Thin Protocol shall support all requested values for negotiation information defined in this document, except the SDU register size which is determined by the dynamic register allocation algorithm (see section 5.8.1.1).

In the Thin Session format used for connection (Connect Request/Ack/Nack), the negotiation information field is used for negotiation between nodes. This field format is described below.

5.11.3.1. Negotiation Information Tag

Negotiation information is composed of a negotiation information tag (NegInf Tag) and a negotiation information value (NegInf Value). NegInf Tag is defined below.

The information marked “M” (Mandatory) shall be contained in the negotiation information field. If any of these mandatory information is not included, the connection shall not be established.

The information marked “O” (Optional) may be contained in the negotiation information field. If the information is not included, a default value is used for each related value.

If unrecognized information is specified, the node shall ignore the information.

<table>
<thead>
<tr>
<th>NegInf Tag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Vendor ID</td>
</tr>
<tr>
<td>01h</td>
<td>Command Set ID</td>
</tr>
<tr>
<td>02h</td>
<td>Transfer Mode</td>
</tr>
<tr>
<td>03h</td>
<td>Reserved</td>
</tr>
<tr>
<td>04h</td>
<td>Idle Timeout (sec)</td>
</tr>
<tr>
<td>05h</td>
<td>Data Transfer Timeout (sec)</td>
</tr>
<tr>
<td>06h-07h</td>
<td>Reserved</td>
</tr>
<tr>
<td>08h</td>
<td>SDU management register address of Initiator node</td>
</tr>
<tr>
<td>09h</td>
<td>SDU management register address of Responder node</td>
</tr>
<tr>
<td>0Ah</td>
<td>SDU register address of Initiator node</td>
</tr>
<tr>
<td>0Bh</td>
<td>SDU register address of Responder node</td>
</tr>
<tr>
<td>0Ch</td>
<td>SDU register size of Initiator node</td>
</tr>
<tr>
<td>0Dh</td>
<td>SDU register size of Responder node</td>
</tr>
<tr>
<td>0Eh - 3Fh</td>
<td>Reserved</td>
</tr>
<tr>
<td>40h - 7Fh</td>
<td>Not defined</td>
</tr>
<tr>
<td>80h - BFh</td>
<td>Vendor Unique Tag</td>
</tr>
<tr>
<td>C0h - FFh</td>
<td>Not defined</td>
</tr>
</tbody>
</table>
5.11.3.2. Negotiation Information Format

In the negotiation information field, some information may be included. Each information field uses the format described below. The information format that has a NegInf Tag value between 40h to 7Fh and C0h to FFh is not defined in this document. If the node receives such values, the connection shall not be established.

5.11.3.2.1. Negotiation Information Format

The information for which the NegInf Tag value is 00h to 3Fh or 80h to BFh uses the following format.

<table>
<thead>
<tr>
<th>NegInf Tag</th>
<th>Length</th>
<th>NegInf Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>06h</td>
<td>Vendor ID</td>
</tr>
</tbody>
</table>

![Figure 31 Negotiation information format - 2](image)

5.11.3.2.2. Vendor ID

<table>
<thead>
<tr>
<th>NegInf Tag (00h)</th>
<th>Length (06h)</th>
<th>Reserved</th>
<th>Vendor ID</th>
</tr>
</thead>
</table>

![Figure 32 Vendor ID](image)

Vendor ID: the same value as Module_Vendor_ID of Configuration ROM.

Reserved field shall be set to 0.

5.11.3.2.3. Command Set ID

<table>
<thead>
<tr>
<th>NegInf Tag (01h)</th>
<th>Length (02h)</th>
<th>Command set ID</th>
<th>Version number</th>
</tr>
</thead>
</table>

![Figure 33 Command set ID](image)

Command set ID: Identifier specified command set. 01h is used for DPP Command Set. The other value is reserved.

Version number: Version number of command set. The value is specified by a command set (e.g. In this document, 10h is used for DPP Command Set version.)
5.11.3.2.4. Transfer Mode

<table>
<thead>
<tr>
<th>NegInf Tag (02h)</th>
<th>Length (02h)</th>
<th>Low-level interface</th>
<th>Transfer mode</th>
</tr>
</thead>
</table>

Figure 34 Transfer mode

Low-level interface: 01h as IEEE1394-1995.
Transfer mode: 00h as Asynchronous PUSH model

5.11.3.2.5. Timeout

The node receiving this tag shall set the timeout value greater than this value.
(i.e. The node issuing this tag shall set this value to the minimum timeout value it wants the connecting node to apply.)

If this information is not included, a default value shall be used as the timeout period.  The default value of each timeout is shown below.

<table>
<thead>
<tr>
<th>Table 6 Default timeout value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
</tr>
<tr>
<td>Idle timeout (04h)</td>
</tr>
<tr>
<td>Data transfer timeout (05h)</td>
</tr>
</tbody>
</table>

Figure 35 Timeout

Timeout value: Timeout value is in seconds.
5.11.3.2.6. Register Space Address

Figure 36  Register space address

Register space address: The address of the register space. The last 2 bits of this field shall set to 0 for quadlet boundary.

5.11.3.2.7. Register Space Size

Figure 37  Register space size

Register space size: Register space size in bytes. This value shall be at least 512 bytes. The last 2 bits of this field shall set to 0 for quadlet boundary. Reserved field shall be set to 0.

5.11.3.2.8. Vendor Unique Tag

Figure 38  Vendor unique tag

This negotiation information is defined by each Vendor that is specified by Vendor ID (described in section 5.11.3.2.2). The usage and value are not defined in this document.
5.11.4. Format used for Connection

5.11.4.1. Connect Request Format

This format shall be used when S_Connect.req is issued by the Application. The Thin Session shall create these fields and issue T_Connect.req to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (10h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfType (40h)</td>
<td>Reserved</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination Machine ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Machine ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Negotiation Information</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 39  Connect request format**

SessionID, MsgType, InfType: Described in section 5.11.2.
Version: Version of the Thin Protocol (10h for this version).
Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.
Destination Machine ID: Machine ID of the node which receives a connect request.
Source Machine ID: Machine ID of the node which sends a connect request.
Negotiation Information: Described in section 5.11.3.
Reserved field shall be set to 00h.
5.11.4.2. Connect Ack Format

This format shall be used when S_Connect.rsp representing a connect confirmation is issued by the Application. The Thin Session shall create these fields and issue T_Connect.rsp to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (11h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfType (40h)</td>
<td>Reserved</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination Machine ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source Machine ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negotiation Information</td>
<td></td>
</tr>
</tbody>
</table>

Figure 40  Connect ack format

SessionID, MsgType, InfType: Described in section 5.11.2.
Version: Version of the Thin Protocol (10h for this version).
Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.
Destination Machine ID: Machine ID of the node which receives a connect ack.
Source Machine ID: Machine ID of the node which sends a connect ack.
Negotiation Information: Described in section 5.11.3.
Reserved field shall be set to 00h.
### 5.11.4.3. Connect Nack Format

This format shall be used when S_Connect.rsp representing a connect rejection is issued by the Application. The Thin Session shall create these fields and issue T_Connect.rsp to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (12h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfType (40h)</td>
<td>Result</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination Machine ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Machine ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negotiation Information</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 41  Connect nack format**

SessionID, MsgType, InfType: Described in section 5.11.2.

Version: Version of the Thin Protocol (10h for this version).

Result: Result of connect rejection.

<table>
<thead>
<tr>
<th>Result Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>01h</td>
<td>Rejected by user</td>
</tr>
<tr>
<td>10h</td>
<td>Already connected&lt;sup&gt;22&lt;/sup&gt;</td>
</tr>
<tr>
<td>11h</td>
<td>No resource for more connection</td>
</tr>
<tr>
<td>12h</td>
<td>Illegal negotiation parameter</td>
</tr>
<tr>
<td>FFh</td>
<td>Unspecified reason</td>
</tr>
</tbody>
</table>

Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.

Destination Machine ID: Machine ID of the node which receives a connect nack.

Source Machine ID: Machine ID of the node which sends a connect nack.

---

<sup>22</sup> When the Responder node returns a connect nack with the result value of 10h (Already connected) but the Requester node does not have the connection, the connection resource of the Responder node might be remain. In that case, the Requester node may issue a disconnect request to release the resource of the Responder node.
Negotiation Information: Described in section 5.11.3. This field in this format is optional. If the node attaches Negotiation information, the Negotiation information value shall be set to an acceptable value of the node. When the node has no acceptable value, all bits in the Negotiation information value field shall be set to 0.

5.11.5. Format used for Reconnection
5.11.5.1. Reconnect Request Format
This format shall be used when T_Error.ind is indicated from the Thin Transaction and a reconnection is necessary. The Thin Session shall create these fields and issue T_Reconnect.req to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (40h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfType (40h)</td>
<td>Reserved</td>
<td>Length (00 10h)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination Machine ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source Machine ID</td>
<td></td>
</tr>
</tbody>
</table>

Figure 42 Reconnect request format

SessionID, MsgType, InfType: Described in section 5.11.2.
Version: Version of the Thin Protocol (10h for this version).
Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.
Destination Machine ID: Machine ID of the node which receives a reconnect request.
Source Machine ID: Machine ID of the node which sends a reconnect request.
Reserved field shall be set to 00h.
5.11.5.2. **Reconnect Ack Format**

This format shall be used when T_Reconnect.ind is indicated from the Thin Transaction and a reconnection is established. The Thin Session shall create these fields and issue T_Reconnect.rsp to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (41h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfType (40h)</td>
<td>Reserved</td>
<td>Length (00 10h)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Destination Machine ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source Machine ID</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 43** Reconnect ack format

SessionID, MsgType, InfType: Described in section 5.11.2.
Version: Version of the Thin Protocol (10h for this version).
Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.
Destination Machine ID: Machine ID of the node which receives a reconnect ack.
Source Machine ID: Machine ID of the node which sends a reconnect ack.
Reserved field shall be set to 00h.
5.11.5.3. Reconnect Nack Format
This format shall be used when T_Reconnect.ind is indicated from the Thin Transaction and a reconnection is rejected. The Thin Session shall create these fields and issue T_Reconnect.rsp to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (42h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfType (40h)</td>
<td>Result</td>
<td>Length (00 10h)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination Machine ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Machine ID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 44 Reconnect nack format

SessionID, MsgType, InfType: Described in section 5.11.2.
Version: Version of the Thin Protocol (10h for this version).
Result: Result of reconnect rejection.

Table 8 Result of reconnect rejection

<table>
<thead>
<tr>
<th>Result Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>20h</td>
<td>Resource is already discarded</td>
</tr>
<tr>
<td>FFh</td>
<td>Unspecified reason</td>
</tr>
</tbody>
</table>

Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.
Destination Machine ID: Machine ID of the node which receives a reconnect nack.
Source Machine ID: Machine ID of the node which sends a reconnect nack.
5.11.6. Format used for Disconnection

5.11.6.1. Disconnect Format

This format shall be used when S_Disconnect.req is issued from the Application. The Thin Session shall create these fields and issue T_Disconnect.req to the Thin Transaction.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (30h)</th>
<th>InfType (00h)</th>
<th>Version (10h)</th>
<th>InfType (40h)</th>
<th>Result</th>
<th>Length (00 10h)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Destination Machine ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Source Machine ID</td>
</tr>
</tbody>
</table>

**Figure 45 Disconnect format**

SessionID, MsgType, InfType: Described in section 5.11.2.
Version: Version of the Thin Protocol (10h for this version).
Result: Result of disconnect.

**Table 9 Result of disconnect**

<table>
<thead>
<tr>
<th>Result Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>01h</td>
<td>Disconnected by user</td>
</tr>
<tr>
<td>02h</td>
<td>1394 connection is disconnected</td>
</tr>
<tr>
<td>FFh</td>
<td>Unspecified reason</td>
</tr>
</tbody>
</table>

Length: Length from the Destination Machine ID field to the last byte of this packet in bytes.
Destination Machine ID: Machine ID of the node which receives a disconnect request.
Source Machine ID: Machine ID of the node which sends a disconnect request.
5.11.7. Format Used for Data Transfer

This format shall be used when S_Command.req or S_Command.rsp is issued from the Application. The Thin Session shall determine whether segmentation is necessary in accordance with the Application data size and the SDU register size, create these fields, and issue T_SDUSend to the Thin Transaction.

5.11.7.1. Application Data Segmentation

The Thin Session segmentation sequence shall use following steps.

i) The Thin Session applies the Application data header (Application data length, Destination Machine ID, Source Machine ID, Destination PID, and Source PID; the meanings of these fields are described in sections 5.11.7.2 and later) to the Application data.

ii) It is determined if the data (Application data with the Application data header) can be sent all at once or not by using the SDU register.

iii) If it can not, the Thin Session segments the data (Application data with the Application data header). If it can, the data is sent by using the Single data format (described in section 5.11.7.2).

iv) Each segment data is sent. The location of the segment data (first/interim/last) is described in the segment flag field.

![Diagram](https://via.placeholder.com/150)

**Application data header**

- Application data length
- Destination Machine ID
- Source Machine ID
- Destination PID
- Source PID
- Application data

- First segment data
- Interim segment data
- Last segment data

**Figure 46 Application data segmentation**
5.11.7.2. Single Data Format

This format shall be used when the Application data is small enough to be transferred without segmentation.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (20h)</th>
<th>InfType (80h)</th>
<th>Segment flag (C0h)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Length</td>
</tr>
<tr>
<td>Sequence number (0000h)</td>
<td>Remaining number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Command type</td>
<td>Command ID</td>
<td></td>
</tr>
<tr>
<td>Application data length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Machine ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Machine ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination PID</td>
<td>Source PID</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 47 Single data format**

Session ID, MsgType, InfType: Described in section 5.11.2.
Segment flag: Segmentation Information flag. It shall be set to C0h in this packet.
Length: Length from the Sequence number field to the last byte of the Application data in bytes.
Sequence number: Sequence number of the segment data. It shall be set to 0000h in this packet.
Remaining number (optional): Remaining number of segment data. If used, it shall be set to 0001h.
If the remaining number field is not used, it shall be set to 0000h.
Command type: Command specified field.

### Table 10 Command type

<table>
<thead>
<tr>
<th>Command type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Command request</td>
</tr>
<tr>
<td>40h</td>
<td>Command response (Ack)</td>
</tr>
<tr>
<td>80h</td>
<td>Command response (Nack)</td>
</tr>
</tbody>
</table>

Command ID: When the Application issues a command request (S_Command.req), the Thin session applies a unique ID number for identifying the Application command. When the Application issues a command response, the Application uses the same ID that was used in the corresponding command request.

Reserved field shall be set to 00h.

Application data length: Length from the Destination Machine ID field to the last byte of the Application data in bytes.

Destination Machine ID: Machine ID of the node which receives a data transfer request.

Source Machine ID: Machine ID of the node which sends a data transfer request.

Destination PID, Source PID: Identification Number to specify the Application program.23

**NOTE:**

The fields from the Application data length field to the Source PID field are treated as the Application data header.

---

23 The Direct Print Command set does not use this field. The field shall be set to 00 00h.
5.11.7.3. Segment Data Format

This format shall be used for transferring a segmented data.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (20h)</th>
<th>InfType (80h)</th>
<th>Segment flag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Sequence number</td>
</tr>
<tr>
<td>Reserved</td>
<td>Command type</td>
<td></td>
<td>Remaining number</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Command ID</td>
</tr>
<tr>
<td>Segment data (First/Interim/Last)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 48 Segment data format

The Application data header is included in the first segment data. The Application data header uses the same format as the header used for the single data format. See 5.11.7.2.

SessionID, MsgType, InfType: Described in section 5.11.2.

Segment flag: Segment information is described in this field. 40h is used for the first segment data, 00h is for all interim segment data, 80h is the last segment data.

<table>
<thead>
<tr>
<th>Segment flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>40h</td>
<td>First segment data</td>
</tr>
<tr>
<td>80h</td>
<td>Last segment data</td>
</tr>
<tr>
<td>00h</td>
<td>Interim segment data</td>
</tr>
</tbody>
</table>

Length: Length from the Sequence number field to the last byte of the Segment data in bytes.

Sequence number: This field describes the sequence number of the segment data. This number starts at 0000h and is increased one by one for each segment data. When the number reaches FFFFh, the next value is started again from 0000h.

Remaining number (optional): This field describes the remaining number of the segment data. This number includes transferring data, so the value of the last segment data is 0001h. If the remaining number of the segment data is more than FFFFh or this field is not used, it shall be set to 0000h.

Command type: Command specified field. See Table 10.

Command ID: This field is the same as the single data format. See 5.11.7.2.
5.11.8. Format used for Command Abort

5.11.8.1. Abort Format

This format shall be used when S_Abort.req is issued from the Application. The Thin Session shall create these fields and issue T_SDUSend.req to the Thin Transaction. If the Thin Transaction is sending an SDU that has the same Command ID as the command abort request, the Thin Session may issue an SDU transfer stop request (T_SDUStop.req) to the Thin Transaction. Otherwise, the Thin Session shall wait to receive SDU Ready (or SDU Reject) before issuing an SDU transfer request.

<table>
<thead>
<tr>
<th>Session ID (00h)</th>
<th>MsgType (21h)</th>
<th>InfType (80h)</th>
<th>Segment flag (C0h)</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence number (0000h)</td>
<td>Remaining number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>Command type</td>
<td>Command ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application data Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination Machine ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Machine ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination PID</td>
<td>Source PID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application data (higher layer abort command)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 49**  Abort format

SessionID, MsgType, InfType: Described in section 5.11.2.
Segment flag: Segmentation Information flag. It shall be set to C0h in this packet.
Length: Length from the Sequence number field to the last byte of the Application data in bytes.
Sequence number: Sequence number of the segment data. It shall be set to 0000h in this packet.
Remaining number (optional): Remaining number of segment data. If used, it shall be set to 0001h.
If the remaining number field is not used, it shall be set to 0000h.
Command type: Command specified field. It shall be set to 00h.
Command ID: Command ID of the Application data that is requested to be aborted.
Application data length: Length from the Destination Machine ID field to the last byte of the Application data in bytes.

Destination Machine ID: Machine ID of the node which receives a command abort request.
Source Machine ID: Machine ID of the node which sends a command abort request.
Destination PID, Source PID: Identification Number to specify the Application program.\textsuperscript{24}

Application data: Higher layer abort command which is sent to another node (e.g. Cancel command of DPP Command set).

\textsuperscript{24} The Direct Print Command set does not use this field. The field shall be set to 00 00h.
5.11.9. **Thin Transaction Service**

The Thin Transaction provides the following services to the Thin Session:

- Connection establishment and allocation of resources \(< T\_Connect (req, ind, rsp, cnf) >\)
- Re-establishment of a connection \(< T\_Reconnect (req, ind, rsp, cnf) >\)
- Disconnect and release of resources \(< T\_Disconnect (req, ind) >\)
- Transfer of data \(< T\_SDUSend (req, ind, rsp, cnf) >\)
- Abort of data transfer \(< T\_SDUStop (req, ind, rsp, cnf) >\)
- Indication of errors \(< T\_Error (ind) >\)

The Thin Transaction operation to perform the services above is described below. The service of indication of errors has no packet, since it is an internal process issued by the Thin Transaction to the Thin Session.

5.11.10. **Register Space**

The three Registers mentioned below shall be used in the Thin Transaction.

- **Connection register**

  The Connection register shall be used for the following operations:
  - Connect Request/Response
  - Reconnect Request/Response
  - Disconnect Request

  ![Connection register space](image)

  **Figure 50** Connection register

- **SDU register**

  The SDU register shall be used for the following operation:
  - SDU Transfer

  ![SDU register space](image)

  **Figure 51** SDU register
SDU management register

The SDU management register contains SDU response space and SDU control space.

SDU response space shall be used for following operations:
- SDU Transfer Ready
- SDU Transfer Reject

SDU control space shall be used for the following operations:
- SDU Transfer Completion
- SDU Transfer Stop

Figure 52  SDU management register
5.11.11. Data Specification Field

In the Connection register and SDU management register, the kind of data stored shall be described in the first 4 bytes. This field is called the Data specification field and the format is shown below. All bits of the Reserved field shall be set to 0.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Reserved</th>
<th>(Length)</th>
</tr>
</thead>
</table>

Figure 53 Data specification field

Data Types: The kind of data stored.

Table 12 Data type defined value

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10h</td>
<td>Connect Req</td>
</tr>
<tr>
<td>11h</td>
<td>Connect Ack</td>
</tr>
<tr>
<td>12h</td>
<td>Connect Nack</td>
</tr>
<tr>
<td>30h</td>
<td>Disconnect Req</td>
</tr>
<tr>
<td>40h</td>
<td>Reconnect Req</td>
</tr>
<tr>
<td>41h</td>
<td>Reconnect Ack</td>
</tr>
<tr>
<td>42h</td>
<td>Reconnect Nack</td>
</tr>
<tr>
<td>E0h</td>
<td>SDU Comp</td>
</tr>
<tr>
<td>E1h</td>
<td>SDU Stop</td>
</tr>
<tr>
<td>F0h</td>
<td>SDU Ready</td>
</tr>
<tr>
<td>F1h</td>
<td>SDU Reject</td>
</tr>
</tbody>
</table>

Length: The length of the data (issued by the Thin Session) is described in bytes. This field shall be used only when accessing the Connection register. For the SDU register and SDU management register access, this field is reserved. The machine which has had data written into the Connection register uses this field to determine if all data have arrived.
5.11.12. Operation used for Connection

5.11.12.1. Connect Request

When T_Connect.req is issued from the Thin Session of the Initiator node, the Thin Transaction shall allocate the SDU register and SDU management register. After allocating the register spaces, the Thin Transaction shall attach the data specification field and send this packet to the Connection register of the Responder node.

When allocation of register spaces cannot be performed, the Thin Transaction shall indicate failure of connection to the Thin Session.

<table>
<thead>
<tr>
<th>Data Type (10h)</th>
<th>Reserved</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data from Thin Session (described in section 5.11.4.1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 54 Connect request

5.11.12.2. Connect Response

When T_Connect.rsp is issued from the Thin Session of the Responder node and it represents a connection confirmation, the Thin Transaction shall allocate the SDU register and SDU management register. After allocating the register spaces, the Thin Transaction shall attach the data specification field and send this packet to the Connection register of the Initiator node.

When T_Connect.rsp represents a connection rejection, the Thin Transaction shall attach the data specification field and send this packet to the Connection register of Initiator node, without allocating register spaces.

<table>
<thead>
<tr>
<th>Data Type (11h/12h)</th>
<th>Reserved</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data from Thin Session (described in section 5.11.4.2 and 5.11.4.3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 55 Connect response
5.11.13. Operation used for Reconnection

5.11.13.1. Reconnect Request

When T_Reconnect.req is issued from the Thin Session of the Sender node, the Thin Transaction shall attach the data specification field and send this packet to the Connection register of the Receiver node.

Connection register

<table>
<thead>
<tr>
<th>Data Type (40h)</th>
<th>Reserved</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data from Thin Session (described in section 5.11.5.1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 56 Reconnect request

5.11.13.2. Reconnect Response

When T_Reconnect.rsp is issued from the Thin Session of the Receiver node, the Thin Transaction shall attach the data specification field and send this packet to the Connection register of the Sender node.

The data type value in the data specification field shall represent Reconnect ack (in case of a reconnect confirmation) or Reconnect nack (in case of a reconnect rejection).

Connection register

<table>
<thead>
<tr>
<th>Data Type (41h/42h)</th>
<th>Reserved</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data from Thin Session (described in section 5.11.5.2 and 5.11.5.3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 57 Reconnect response
5.11.14. Operation used for Disconnection

5.11.14.1. Disconnect Request
When T_DISCONNECT:req is issued from the Thin Session of the Sender node, the Thin Transaction shall release the allocated register, attach the data specification field and send this packet to the Connection register of the Receiver node.

Connection register

```
<table>
<thead>
<tr>
<th>Data Type (30h)</th>
<th>Reserved</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data from Thin Session (described in section 5.11.6.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 58 Disconnect request

5.11.15. Operation used for SDU Transfer

5.11.15.1. SDU Transfer
When T_SDUSEND is issued from the Thin Session of the Sender node, the Thin Transaction shall send the data from the Thin Session as a packet to the SDU register of the Receiver node without applying any field.

SDU register

```
<table>
<thead>
<tr>
<th>Data Type (E0h)</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Data from Thin Session (described in section 5.11.7 and later)</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 59 SDU transfer

5.11.15.2. SDU Transfer Completion
When the transfer of an SDU is completed, the Thin Transaction shall send the data specification field representing SDU transfer completion (SDU Comp) to the SDU Control Space in the SDU management register of the Receiver node.

SDU management register (SDU control Space)

```
<table>
<thead>
<tr>
<th>Data Type (E0h)</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 60 SDU transfer completion
5.11.15.3. SDU Transfer Stop

When T_SDUSTop is issued from the Thin Session of the Sender node, the Thin Transaction shall send the data specification field representing SDU transfer stop (SDU Stop) to the SDU control space in the SDU management register of the Receiver node.

<table>
<thead>
<tr>
<th>SDU management Register (SDU control space)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type (E1h)</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
</tbody>
</table>

Figure 61 SDU transfer stop

5.11.15.4. SDU Transfer Ready

When SDU Comp is stored in the SDU control space, the Thin Transaction shall issue an SDU transfer indication (T_SDUSend.ind) to the Thin Session that data is stored. After the Thin Session stores the data to the buffer and makes the SDU register ready for receiving next SDU, the Thin Session issues an SDU transfer response (T_SDUSend.rsp) that indicates the SDU register is available for writing.

When the Thin Transaction receives the service, the Thin Transaction shall send the data specification field representing SDU transfer ready (SDU Ready) to the SDU response space in the SDU management register of the Sender node.

After SDU Ready is stored from the Receiver node, the Thin Transaction of the Sender node shall issue an SDU transfer confirmation (T_SDUSend.cnf) to the Thin Session. The Thin Session of Sender node may transfer the next packet at this timing.

<table>
<thead>
<tr>
<th>SDU management register (SDU response space)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type (F0h)</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
</tbody>
</table>

Figure 62 SDU transfer ready
5.11.15.5. SDU Transfer Reject

When SDU Comp is stored in SDU control space, the Thin Transaction shall issue an SDU transfer indication (T_SDUSend.ind) to the Thin Session that data is stored. When the Thin Session cannot store the data to the buffer, the Thin Session shall make the SDU register ready for receiving next SDU and issue an SDU transfer response (T_SDUSend.rsp) to the Thin Transaction that indicates the SDU transfer is rejected.

When the Thin Transaction receives the service, the Thin Transaction shall send the data specification field representing SDU transfer reject (SDU Reject) to the SDU response space in the SDU management register of the Sender node.

After SDU Reject is stored from the Receiver node, the Thin Transaction of the Sender shall issue an SDU transfer confirmation (T_SDUSend.cnf) to the Thin Session that indicates the SDU transfer is rejected. The Thin Session may retry to send the same SDU. If the Thin Session does not retry, it shall issue an error indication to the Application. When an error is indicated, the Application shall issue a command abort request.

**SDU management register (SDU response space)**

<table>
<thead>
<tr>
<th>Data Type (F1h)</th>
<th>Reserved</th>
</tr>
</thead>
</table>

**Figure 63**  SDU transfer reject
6. Direct Print Command Set
6.1. Overview
6.1.1. Command Model

This command set employs a push model in which the image source device mainly controls the target device.

Pull models in which the target devices mainly control the image source devices is out of the scope for this version of the Direct Print Command set.

![Diagram of push model Direct Print](image_url)

Figure 64 Push model Direct Print
6.1.2. Command Sequence

The direct print command sequence is made up of two main portions.

The basic sequence is performed in the following order:

Step 1: Negotiation (Query parameters, and Set parameters)
Step 2: Send data (Send image data)
6.1.3. Negotiation

Since direct print parameters defined in this command set have mandatory default Values, command level negotiation is not required in case the DPP output sequence uses these default Values. On the other hand, in order to utilize the full direct-print capabilities of both the image source device and the target device, command level negotiations will be necessary to determine and set suitable direct print parameters.

![Diagram of Negotiation between Image Source Device and Target Device]

Figure 66 Negotiation between image source device and target device

6.2. Requirements

This command set is most suited to be the upper layer of the Thin Protocol, but other transports are acceptable. A symmetrical and reliable transport is required as the lower layer of this command set, and at least one transport connection between the image source device and the target device is required.
6.3. **Command Set Categorization**

6.3.1. **Command Set Components**

The components that make up the command set are commands, responses, command parameters, and response parameters. A command is considered complete when the issuing device receives a response, unless otherwise noted.

![Figure 67 Command Set Components](image)

6.3.2. **Command Categorization**

Defined commands are composed of the following three types of command.

<table>
<thead>
<tr>
<th>Table 13</th>
<th>Command categorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td></td>
</tr>
<tr>
<td>(1)Generic command:</td>
<td>DPP defined commands that are used for purposes other than negotiation. Specific parameters are defined for commands and corresponding responses</td>
</tr>
<tr>
<td>(2)Negotiation command:</td>
<td>DPP defined commands used for negotiation. Item and Value are used for the parameters that are common to every negotiation command and response.</td>
</tr>
<tr>
<td>(3)Vendor Unique command:</td>
<td>Commands not defined in DPP.</td>
</tr>
</tbody>
</table>
Any command(s) other than the Generic Commands and Negotiation Commands specified in this document is (are) regarded as vendor unique Command(s). A DPP device shall respond with an error to any unknown command such as unknown vendor unique commands.

The following table shows the different packet formats defined in this document.

<table>
<thead>
<tr>
<th>Command command:</th>
<th>Command/Response packet format</th>
<th>Parameter packet format</th>
<th>Parameter contents format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic command:</td>
<td>Generic Command/Response Packet format</td>
<td>Generic Command/Response Parameter format</td>
<td>Command dependent</td>
</tr>
<tr>
<td>Negotiation command:</td>
<td>Negotiation Command/Response Packet format</td>
<td>Item Unit format</td>
<td>Value Content Block format</td>
</tr>
</tbody>
</table>

The packet formats shall be used in the following packet encapsulation hierarchy.

![Command and Response Packet Encapsulation Hierarchy Diagram](image)

Figure 68 Command and Response Packet Encapsulation Hierarchy
6.3.3. Command List

Table 15 shows the list of commands defined in this specification.

<table>
<thead>
<tr>
<th>Category</th>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiation</td>
<td>GetQueryItem</td>
<td>Querying the capabilities of target device</td>
</tr>
<tr>
<td>command</td>
<td>Negotiation</td>
<td>The command used for negotiation. Parameter and response of</td>
</tr>
<tr>
<td></td>
<td>Negotiation</td>
<td>negotiation command employ Item and Value.</td>
</tr>
<tr>
<td></td>
<td>SetQueryItem</td>
<td>Setting the actual capability condition of target device</td>
</tr>
<tr>
<td>Generic</td>
<td>Send²⁵</td>
<td>Sending 1 image data to target device</td>
</tr>
<tr>
<td>command</td>
<td>Cancel</td>
<td>Canceling the previous command being executed</td>
</tr>
<tr>
<td></td>
<td>GetStatus</td>
<td>Obtaining the status of target device</td>
</tr>
<tr>
<td></td>
<td>SendStatus</td>
<td>Sending the status of target device</td>
</tr>
</tbody>
</table>

²⁵ In the case of a printer implementation, this command may be considered as a basis for print execution triggering.
6.4. Negotiation Command

The two commands used for negotiation are:

- **GetQueryItem Command**
- **SetQueryItem Command**

The same commands are used for negotiation of all items.

Commands and command parameters used to send and receive Item and Value information in negotiation commands are designed to use the packet format shown below. The command and command parameter packet format structures are common between command and responses.

The command/response parameter for negotiation commands is called Query Unit List and consists of Item and Value information.

![Figure 69 Negotiation Command and Command Parameters](image-url)
Parameters for commands, and return values for responses of negotiation commands employ an Item and Value structure.

Table 16  
Item and Value

| Item:  | Category of parameter group negotiable between image source device and target device.  
|        | ex) ImageSize |
| Value: | Each parameter of Item representing a specific capability. Value is subordinated to Item.  
|        | ex) VGA, XGA, … |

In the case of a negotiation command, the image source device and target device play the roles of command issuing device and command responding device, respectively. Item and Value are defined to be common to all negotiation commands.

6.4.1. Implementation Level in Negotiation Command

Command, Item and Value of negotiation command has the following implementation levels.

6.4.1.1. Implementation Level of Negotiation Command

DPP devices shall implement DPP defined commands as the following table indicates, according to whether the device is an image source device or a target device.

Table 17  
Implementation level of Negotiation Command

<table>
<thead>
<tr>
<th>Image source device:</th>
<th>May select and use the necessary command from the DPP defined commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target device:</td>
<td>Shall understand all the DPP defined commands specified in this document and shall respond with the actual condition of implemented capability</td>
</tr>
</tbody>
</table>
### 6.4.1.2. Implementation Level of Item

All Items specified in this document are “Basic Items”.

A DPP device shall understand the meaning of Basic Item and shall respond with the actual condition of implemented capability. An image source device may query a target device about (an) Item(s), and the target device shall respond to the image source device with the Value(s) representing the capability of the queried Item(s).

<table>
<thead>
<tr>
<th>Table 18</th>
<th>Implementation level of Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image source device:</strong></td>
<td>May select and use the necessary command out of Basic Item.</td>
</tr>
<tr>
<td><strong>Target device:</strong></td>
<td>Shall understand all the Basic Items specified in this document and shall respond with the actual condition of implemented capability of that Item.</td>
</tr>
</tbody>
</table>

All Items other than the Basic Items specified in this document are regarded as vendor unique Items. A DPP target device shall respond with “Not supported Item” (See 6.4.4.1) to a command which includes unknown Items such as unknown vendor unique Items.
6.4.1.3. Implementation Level of Value

Value has the following two implementation levels.

Table 19 Implementation level of Value

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Mandatory Value</td>
<td>Every DPP target device shall be capable of supporting Mandatory Values. DPP target devices shall include the Mandatory Value of the corresponding Item in the response to the Item query from the image source device. The Mandatory Value is a default setting for DPP devices, so the target device shall use this Value if a specific Value for an Item is not set by the image source device.</td>
</tr>
<tr>
<td>(2) Optional Value</td>
<td>Implementation of Optional Values defined in this command set is optional to the target device. Target devices shall respond with corresponding Values that are supported, when responding to a query command.</td>
</tr>
</tbody>
</table>

Vendor unique Value(s) subordinated to Basic Items is (are) available. DPP image source devices shall ignore any unknown Value such as unknown vendor unique Value. DPP target devices shall respond with “Not supported Value” (See 6.4.4.2) to a command which includes an unknown Value such as unknown vendor unique Value. Vendor unique Values that have the same meanings as Mandatory and/or Optional Values specified in this document shall be avoided to ensure interoperability.
### 6.4.2. Item/Value Listing
The table below shows the Items and Values described in this document.

<table>
<thead>
<tr>
<th>Table 20 Item/Value list</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Implementation Level</th>
<th>Meaning</th>
<th>Value</th>
<th>Implementation Level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ImageFormat</strong></td>
<td>Basic</td>
<td>Image data format</td>
<td>rawRGB</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>D-rawRGB</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Exif</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>JFIF</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>ImageSize</strong></td>
<td>Basic</td>
<td>Image data size</td>
<td>VGA</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SVGA</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XGA</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SXGA960</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SXGA1024</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>XY</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>maxXY</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>Output Orientation</strong></td>
<td>Basic</td>
<td>Output direction</td>
<td>Device</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dependent</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Portrait</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Landscape</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mirrored-Portrait</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mirrored-Landscape</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
</tbody>
</table>
Continued from previous page

<table>
<thead>
<tr>
<th>Item</th>
<th>Implementation Level</th>
<th>Meaning</th>
<th>Value</th>
<th>Implementation Level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sizing</strong></td>
<td>Basic</td>
<td>Sizing the image within the output media</td>
<td>Device</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dependent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Small</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Medium</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Large</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>PosX</strong></td>
<td>Basic</td>
<td>Position of image within the output media</td>
<td>Device</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dependent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Left</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Center</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Right</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>PosY</strong></td>
<td>Basic</td>
<td>Position of image within the output media</td>
<td>Device</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dependent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Top</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Center</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bottom</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>NumPics</strong></td>
<td>Basic</td>
<td>Number of images per output media</td>
<td>1</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2-255</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>NumCopies</strong></td>
<td>Basic</td>
<td>Number of output media</td>
<td>1</td>
<td>Mandatory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2-65535</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
<tr>
<td><strong>VendorUnique Item</strong></td>
<td>Basic</td>
<td>vendor unique Item</td>
<td>(xxxxx)</td>
<td>Vendor Unique</td>
</tr>
</tbody>
</table>

(Get only)
6.4.3. Negotiation Command/Response Packet Format

All commands and responses share a common packet format, as shown below.

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>OPT</td>
<td>V</td>
<td>D</td>
<td>L</td>
<td>CmdCode</td>
<td>Count</td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Length32 (present when F=1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Command Packet Contents

Figure 70 Negotiation Command/Response Packet Format

F: Length Format

0b = Short format

The valid packet length for this command/response packet will be shown in the 8 bit Length field. The Length32 field shall not be present in this case.

1b = Long format

The valid packet length for this command/response packet will be shown in the 32 bit Length32 field. The Length32 field shall be present in this case.

OPT: Optional Attributes

The meaning of this field is different between commands and responses.

In case of a command, this field acts as a Req field.

Req: Requested (queried) number of Items

0h = Number of Items specified in the Count field.

1h = All Basic Items

2h = All vendor unique Items

3h = All available Items

In case of a response, this field acts as a Status field.

Status: Status response

0h = No error

1h = Error
V: Vendor Unique Flag
   Negotiation commands shall set this field to 0.
   0b = DPP defined Commands
   1b = Vendor Unique Commands

D: Direction
   0b = Command packet
   1b = Response packet

L: Level
   0b = Command/Response Level
   Commands and Responses shall set this field to 0.

CmdCode: Command code
   Values are dependent on each command.

Count: Content count
   The number of command packet content blocks following this field.

Length: Packet Length
   The length of this Command/Response packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 255 bytes. Command and Response packets with a packet length exceeding 255 bytes shall set the F field to 1, and use the Length32 field.

Length32: Packet Length32
   The length of this Command/Response packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 4,294,967,295 bytes. When this field is used, the value of the Length field is invalid. Command and Response packets with a packet length less than 255 bytes shall set the F field to 0, and use the Length field.

Command Packet Contents: Contents
   The Query Unit List shall be encapsulated in this field.
6.4.4. Query Unit List Format

The information included in the Query Unit List will differ between GetQueryItem, and SetQueryItem commands, and between the command and response.

* In case of the GetQueryItem command;
  The command Query Unit Lists will consist of Item information to be queried.
  The response Query Unit Lists will consist of Item information queried, and Value information for each Item queried.

* In case of the SetQueryItem command;
  The command Query Unit Lists will consist of Item information to be set, and Value information for each Item to be set.
  The response Query Unit Lists may consist of Item information to be set, and result Value information for each Item.

A Query Unit List is comprised of 1 or more Item Units.
An Item Unit follows the Item Unit format, and is comprised of Item information, and corresponding Value Content Blocks (when necessary).
A Value Content Block follows the ValueContent format, and is comprised of Value information.

Figure 71 Negotiation Command Parameters – Query Unit List
Since multiple Item information and multiple Value information are allowed in one Query Unit List, the Query Unit List may actually be a list consisting of multiple Item Units.

Figure 72 Negotiation Command Parameters – Query Unit List
6.4.4.1. Item Unit Format

All Item Units share a common packet format, as shown below.

(1) DPP Defined Items (V = 0)

<table>
<thead>
<tr>
<th>F</th>
<th>OPT</th>
<th>V</th>
<th>D</th>
<th>L.</th>
<th>ItemCode</th>
<th>Count</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 73 Item Unit Format of DPP Defined Items

F: Length Format

0b = Short format
The valid packet length for this Item Unit packet will be shown in the 8 bit Length field. The Length32 field shall not be present in this case.

1b = Long format
The valid packet length for this Item Unit packet will be shown in the 32 bit Length32 field. The Length32 field shall be present in this case.

OPT: Optional Attributes
The meaning of this field is different between commands and responses.
In case of a command, this field is unused and shall be filled with 0.
In case of a response, this field acts as a Status field.

Status: Status response
0h = No error
1h = Error
2h = Not supported Item

V: Vendor Unique Flag
DPP defined Items shall set this field to 0.
0b = DPP defined Items
1b = Vendor unique Items
D: Direction

0b = Command packet
1b = Response packet

L: Level

1b = Parameter level
Item Units shall set this field to 1.

ItemCode: Item code
This field will specify a Basic Item.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning (Item)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1h</td>
<td>VendorUniqueItem</td>
</tr>
<tr>
<td>2h</td>
<td>ImageFormat</td>
</tr>
<tr>
<td>3h</td>
<td>ImageSize</td>
</tr>
<tr>
<td>4h</td>
<td>OutputOrientation</td>
</tr>
<tr>
<td>5h</td>
<td>Sizing</td>
</tr>
<tr>
<td>6h</td>
<td>PosX</td>
</tr>
<tr>
<td>7h</td>
<td>PosY</td>
</tr>
<tr>
<td>8h</td>
<td>NumPics</td>
</tr>
<tr>
<td>9h</td>
<td>NumCopies</td>
</tr>
</tbody>
</table>

Count: Content count
The number of Value Content Blocks following this field.

Length: Packet Length
The length of this Item Unit packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 255 bytes. Command and Response packets with a packet length exceeding 255 bytes shall set the F field to 1, and use the Length32 field.
Length32: Packet Length32

The length of this Item Unit packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 4,294,967,295 bytes. When this field is used, the value of the Length field is invalid. Command and Response packets with a packet length less than 255 bytes shall set the F field to 0, and use the Length field.

Value Contents: Contents

The ValueContent shall be encapsulated in this field.

(2) Vendor unique Items (V = 1)

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>OPT</td>
<td>V</td>
<td>D</td>
<td>L</td>
<td>ItemCode</td>
<td>Count</td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Length32 (present when F=1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CompanyID</td>
<td>CompAttrib</td>
<td></td>
</tr>
</tbody>
</table>

Figure 74: Item Unit Format of Vendor unique Items

ItemCode: Item Code

Vendor unique Item codes shall be specified in byte values

CompanyID: Company ID

This 24 bit company ID value provides the company ID of the vendor defining the particular vendor unique command/response. The company ID value is uniquely assigned to each vendor by the IEEE.

CompAttrib: Company Attribute

The definition of this 8 bit value is dependent on the vendor represented by the company ID field.

Usage of all other fields shall be identical to DPP defined Items.
6.4.4.2. Value Content Block Format

(1) DPP Defined Values (V = 0)

<table>
<thead>
<tr>
<th>F</th>
<th>OPT</th>
<th>V</th>
<th>D</th>
<th>L</th>
<th>ValueCode</th>
<th>Count</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 75 Value Content Block Format of DPP Defined Values

F: Length Format

0b = Short format

The valid packet length for this Value Content Block packet will be shown in the 8 bit Length field. The Length32 field shall not be present in this case.

1b = Long format

The valid packet length for this Value Content Block packet will be shown in the 32 bit Length32 field. The Length32 field shall be present in this case.

OPT: Optional Attributes

The meaning of this field is different between commands and responses.

In case of a command, this field is unused and shall be filled with 0.

In case of a response, this field acts as a Status field.

Status: Status response

0h = No error

1h = Error

2h = Not supported Value

V: Vendor Unique Flag

DPP defined Values shall set this field to 0.

0b = DPP defined Values

1b = Vendor unique Values
D: Direction
   0b = Command packet
   1b = Response packet

L: Level
   10b = Parameter contents level
   Value Content Blocks shall set this field to 10b.

ValueCode: Value code
   The meaning of this field is dependent on the Item.

Count: Content count
   The number of Value contents following this field.

Length: Packet Length
   The length of this Value Content Block packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 255 bytes. Command and Response packets with a packet length exceeding 255 bytes shall set the F field to 1, and use the Length32 field.

Length32: Packet Length32
   The length of this Value Content Block packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 4,294,967,295 bytes. When this field is used, the value of the Length field is invalid. Command and Response packets with a packet length less than 255 bytes shall set the F field to 0, and use the Length field.

Value Contents: Contents
   The meaning of this field is dependent on the Item.
(2) Vendor unique Values (V = 1)

ValueCode: Value Code

Vendor unique Value codes shall be specified in byte values.

CompanyID: Company ID

This 24 bit company ID value provides the company ID of the vendor defining the particular vendor unique command/response. The company ID value is uniquely assigned to each vendor by the IEEE.

CompAttrib: Company Attribute

The definition of this 8 bit value is dependent on the vendor represented by the company ID field.

Usage of all other fields shall be identical to DPP defined Items.
6.4.5. Negotiation Command Details

There are two commands that comprise the Negotiation command set. This section will describe details of each of these commands.

**Cmd Code:**

Defines the Cmd Code field of the command/response packet.

**Function:**

Describes the functionality of the command.

**Command Parameters:**

Defines the Command Packet Contents field of the command packet.

**Return Parameters:**

Defines the Command Packet Contents field of the response packet.

**Command/Response Sequence:**

Describes the command/response transaction.
6.4.5.1. **GetQueryItem**

**Cmd Code:**

01h

**Function:**

This command is a request for information on Items specified on a Query unit basis.

**Command Parameters:**

Query Unit List: The Query Unit List for this command will include Item Units specifying the Item names subject to the information request.

**Return Parameters:**

Query Unit List: The Query Unit List for this response will include Item Units of the Items specified, and Value Content Blocks for corresponding Items in the format specified when the command was issued.

**Command/Response Sequence:**

CMD: `GetQueryItem`

PARA: Query Unit List

RESP: `GetQueryItem` Response

PARA: Query Unit List

**Usage:**

**CASE 1: Query Items specified**

The command Query Unit List shall be comprised of one or more Item Units specifying the Item.

The response Query Unit List shall be comprised of one or more Item Units specifying the Item, and Value Content Blocks with query result Values.

**CASE 2: Query all Items, all basic Items, or all vendor unique Items**

The command will not require a Query Unit List and the Req field of the command shall be 1h, 2h, or 3h.

The response Query Unit List shall be comprised of Item Units for all specified Items with query result Values.
6.4.5.2. SetQueryItem

Cmd Code:

02h

Function:
This command will set Values of Items (1 Value per Item) which are specified by the command parameter. Values set by this command shall be valid until they are re-set by this command, or until the lower layer transport session is disconnected.

Command Parameters:
Query Unit List: The Query Unit List for this command will include Item Units specifying the Item names and corresponding Values subject to Value setting.

In case the Query Unit List consists of multiple Item Units (setting multiple Items in one SetQueryItem command), the setting order of the Items shall be prioritized by the order in which the Items appear in the Query Unit List.

Upon an error, the remaining Items appearing after the error Item are not set.

Return Parameters:
Query Unit List: The Query Unit List for this response will include response information on the Value setting specified when the command was issued, and shall include Item Units of Items that were not successfully set.

Command/Response Sequence:
CMD: SetQueryItem
PARA: Query Unit List

RESP: SetQueryItem Response
PARA: Query Unit List
6.4.6. Negotiation Item/Value Details

The following sections describe details of the basic negotiation Items, and their defined Values. Values for each Item will be set to a default Value when the lower level transport layer initiates and establishes a connection. The default Value for each Item is the Mandatory Value defined for that Item.

ItemCode:
Defines the ItemCode field of the Item Unit packet.

Item Definition:
Describes the Item.

Value Data:
Explains Mandatory Values for each Item, Optional Values for each Item, and defines the ValueCode and Value contents field of the Value Content Block packet.

Vendor unique Value Data:
Explains vendor unique Values for this Item.
6.4.6.1. **VendorUniqueItem**

**Item Code:**

01h

**Item definition:**

Vendor unique Item supported by the target device. (Supported vendor unique Item(s) when used with the *GetQueryItem* command.)

**Value Data:**

No Mandatory Value is defined for this Item.

Vendor unique Items shall be defined as byte values. This value concatenated with the company ID shall address a specific vendor unique Item. The definition and Value formats of the vendor unique Items are beyond the scope of this document.

Values present represent a supported Vendor unique Item in the case of a *GetQueryItem* response. There will be no Values present in the case that there are no vendor unique Items supported. In this case, the Item Unit status of “not supported” shall be returned. A *SetQueryItem* command does not exist for this Item.

<table>
<thead>
<tr>
<th>24</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>VendorUniqueItem</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 77** Value Content Format of **VendorUniqueItem**

**CompanyID**: This 24 bit company ID value provides the company ID of the vendor defining the particular vendor unique Item. The company ID value is uniquely assigned to each vendor by the IEEE.

**CompAttrib**: The definition of this 8 bit value is dependent on the vendor represented by the company ID field.

**VendorUniqueItem**: Vendor unique Item value in byte code.

**Vendor unique Value Data:**

No vendor unique Values are defined for this Item.
6.4.6.2. **ImageFormat**

**Item Code:**

02h

**Item definition:**

Transfer image data format. (Supported image data format(s) when used with the `GetQueryItem` command, and image data format to be set, when used with the `SetQueryItem` command.)

**Value Data (v=0, ValueCode=1h):**

The `rawRGB` format is subject to mandatory support by the target device. The corresponding bit shall always be set in the case of a `GetQueryItem` response, and the bit shall be set when specifying this Value in a `SetQueryItem` command.

When set, the following bits represent supported Values in the case of a `GetQueryItem` response, and the bits specify a Value to be set in the case of a `SetQueryItem` command.

<table>
<thead>
<tr>
<th>I0</th>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>Reserved</th>
</tr>
</thead>
</table>

**Figure 78 Value Content Format of ImageFormat**

- **I0 (mandatory):** `rawRGB` – RGB raw data with sRGB color space.
- **I1:** `D-rawRGB` – RGB raw data with sRGB color space accompanied by dummy data for quadlet allocation
- **I2:** `Exif`
- **I3:** `JFIF`

Refer to the ANNEX section for details on data format and requirements for each format.

**Vendor unique Value Data (v=1, ValueCode =vendor dependent):**

Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.
6.4.6.3. ImageSize

Item Code:
03h

Item definition:
Transfer image size. (Supported image size(s) when used with the GetQueryItem command, and image size to be set, when used with the SetQueryItem command.)

Value Data (v=0, ValueCode depends on data type):
VGA (640*480 pixels) size is subject to mandatory support by the target device. The corresponding bit shall always be set in the case of a GetQueryItem response, and this bit shall be set when specifying this Value in a SetQueryItem command.

Fixed size (v=0, ValueCode=1h):
When set, the following bits represent supported Values in the case of a GetQueryItem response, and the bits specify a Value to be set in the case of a SetQueryItem command.

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>X</th>
<th>S1</th>
<th>S2</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>V (mandatory): VGA 640 * 480 pixels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S: SVGA 800 * 600 pixels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X: XGA 1024 * 768 pixels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1: SXGA960 1280 * 960 pixels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2: SXGA1024 1280 * 1024 pixels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 79 Value Content Format of ImageSize (Fixed Size)
Fixed numerical size (v=0,ValueCode=2h):
A fixed numerical size shall be represented in an X*Y pixels format, as a set of word values representing X immediately followed by Y. Values present represent a supported size value in the case of a GetQueryItem response, and the size values specify values to be set in the case of a SetQueryItem command.

```
16 16
| X Value | Y Value |
```

Figure 80   Value Content Format of ImageSize (Fixed Numerical Size)

X Value : X pixels (hex values)...maximum=65,535 pixels
Y Value : Y pixels (hex values)...maximum=65,535 pixels

MaxXY numerical size (v=0,ValueCode=3h):
A MaxXY numerical size will represent the maximum allowable size the target device can handle, in a maxX * maxY pixels format, as a set of word values representing X immediately followed by Y. Values present represent the supported maximum size value in the case of a GetQueryItem response. A SetQueryItem command does not exist for this Item.

```
16 16
| maxX Value | maxY Value |
```

Figure 81   Value Content Format of ImageSize (MaxXY Numerical Size)

maxX Value: maxX pixels (hex values)...maximum=65,535 pixels
maxY Value: maxY pixels (hex values)...maximum=65,535 pixels

Vendor unique Value Data (v=1 ValueCode/vendor dependent):
Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.
6.4.6.4. **OutputOrientation**

**Item Code:**

04h

**Item definition:**

Output orientation of image. (Supported orientation(s) when used with the \textit{GetQueryItem} command, and orientation to be set, when used with the \textit{SetQueryItem} command.)

**Value Data (v=0, ValueCode=1h):**

DeviceDependent (the output orientation will be decided by the target device) is subject to mandatory support by the target device. The corresponding bit shall always be set in the case of a \textit{GetQueryItem} response, and this bit shall be set when specifying this Value in a \textit{SetQueryItem} command.

When set, the following bits represent supported Values in the case of a \textit{GetQueryItem} response, and the bits specify a Value to be set in case of a \textit{SetQueryItem} command.

<table>
<thead>
<tr>
<th>D</th>
<th>P</th>
<th>L</th>
<th>MP</th>
<th>ML</th>
<th>Reserved</th>
</tr>
</thead>
</table>

**Figure 82 Value Content Format of OutputOrientation**

D (mandatory) : DeviceDependent

P : Portrait

The raster direction of the image data corresponds to the short dimension of the output media

L : Landscape

The raster direction of the image data corresponds to the long dimension of the output media

MP : Mirrored-Portrait

The horizontally mirrored image of Portrait

ML : Mirrored-Landscape

The horizontally mirrored image of Landscape

**Vendor unique Value Data (v=1, ValueCode=vendor dependent):**

Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.
6.4.6.5. Sizing

Item Code:

05h

Item definition:

Output sizing of the image. (Supported sizing(s) when used with the *GetQueryItem* command, and the sizing to be set, when used with the *SetQueryItem* command.)

Value Data (v=0, ValueCode=1h):

*DeviceDependent* (the sizing will be decided by the target device) is subject to mandatory support by the target device. The corresponding bit shall always be set in case of a *GetQueryItem* response, and this bit shall be set when specifying this Value in a *SetQueryItem* command.

When set, the following bits represent a supported Values in the case of a *GetQueryItem* response, and the bits specify a Value to be set in the case of a *SetQueryItem* command.

```
D | S | M | L | Reserved
---|---|---|---|----------
```

**Figure 83  Value Content Format of Sizing**

D (mandatory) : *DeviceDependent*

S   : *Small*

The image output will be sized small, relative to the “M” and “L” sizing within the output media.

M   : *Medium*

The image output will be sized smaller than the “L” sizing and larger than the “S” sizing within the output media.

L   : *Large*

The image output will be sized to fully fit the output orientation without changes to aspect ratio of the image.

Vendor unique Value Data (v=1, ValueCode =vendor dependent):

Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.
6.4.6.6. PosX

Item Code:

06h

Item definition:

Output X direction (the shorter dimension of the media) positioning of the image. (Supported positioning(s) when used with the GetQueryItem command, and position to be set, when used with the SetQueryItem command.)

Value Data (v=0, ValueCode=1h):

DeviceDependent (the sizing will be decided by the target device) is subject to mandatory support by the target device. The corresponding bit shall always be set in the case of a GetQueryItem response, and this bit shall be set when specifying this Value in a SetQueryItem command.

When set, the following bits represent supported Values in the case of a GetQueryItem response, and the bits specify a Value to be set in the case of a SetQueryItem command.

Vendor unique Value Data (v=1, ValueCode =vendor dependent):

Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>L</th>
<th>R</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>D (mandatory) : DeviceDependent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C : Center</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The image output will be positioned at the center in the X direction.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L : Left</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The image output will be positioned towards the left side in the X direction.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R : Right</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The image output will be positioned towards the right side in the X direction.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 84 Value Content Format of PosX
6.4.6.7. *PosY*

**Item Code:**

07h

**Item definition:**

Output Y direction *(the longer dimension of the media)* positioning of the image.
(Supported positioning(s) when used with the `GetQueryItem` command, and position to be set, when used with the `SetQueryItem` command.)

**Value Data (v=0, ValueCode=1h):**

*DeviceDependent* (the sizing will be decided by the target device) is subject to mandatory support by the target device. The corresponding bit shall always be set in the case of a `GetQueryItem` response, and this bit shall be set when specifying this Value in a `SetQueryItem` command.

When set, the following bits represent supported Values in the case of a `GetQueryItem` response, and the bits specify a Value to be set in the case of a `SetQueryItem` command.

```
+---+---+---+---+---+---+---+---+
| D | C | T | B |     |     |     | Reserved |
+---+---+---+---+---+---+---+---+
```

**Figure 85** Value Content Format of *PosY*

- **D** (mandatory) : *DeviceDependent*
- **C** : *Center*
  - The image output will be positioned at the center in the Y direction.
- **T** : *Top*
  - The image output will be positioned towards the top in the Y direction.
- **B** : *Bottom*
  - The image output will be positioned towards the bottom in the Y direction.

**Vendor unique Value Data (v=1, ValueCode =vendor dependent):**

Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.
6.4.6.8. *NumPics*

**Item Code:**

08h

**Item definition:**

Number of images laid out per output unit (page.) (Supported quantity of images laid out (number of frames) per output unit when used with the *GetQueryItem* command, and the number of images to be laid out (number of frames) per output unit, when used with the *SetQueryItem* command.)

**NOTE:**

The Value of this Item does not represent the number of valid image data sets to be actually transferred, but is the number of images the target device assumes to lay out in one output unit.

*Image transfer(s) to a target device for one output unit is assumed complete when the image transfer command Send is issued an equal number of times as the Value set for this Item, or a Send command with a image data length of zero. For example, when the target device supports a NumPics Value of 9, and the Value of this Item is set to 9, the target device will either expect a Send command 9 times, or a Send command with an image data length of 0, as a completion of the image transfer for the output.*

**Value Data (v=0, ValueCode=1h):**

A minimum number of 1 is subject to mandatory support by the target device.

A fixed numerical quantity shall be represented as a byte value. Values present represent the supported quantity of images that can be laid out (number of frames) per output unit in the case of a *GetQueryItem* response, and the value specifies the quantity of images laid out (number of frames) per output unit in the case of a *SetQueryItem* command. This value field shall be aligned in quadlet boundary and filled with zeros.

```
8  8  8  8

<table>
<thead>
<tr>
<th>NumPics(1)</th>
<th>NumPics(2)</th>
<th></th>
<th>NumPics(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Figure 86 Value Content Format of NumPics**

*NumPics*: Number of images laid out per output unit

1 - 255 images (1 is mandatory)
Vendor unique Value Data (v=1, ValueCode = vendor dependent):
Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.

6.4.6.9. **NumCopies**

**Item I:**

09h

**Item definition:**
Number of units (pages) of the output. (Supported maximum number of units when used with the GetQueryItem command, and the specified number of units of the output unit, when used with the SetQueryItem command.)

**Value Data (v=0, ValueCode=1h):**

A minimum number of 1 is subject to mandatory support by the target device.
A fixed numerical quantity shall be represented as a byte value. Values present represent the supported maximum number of units of the output in the case of a GetQueryItem response, and the value specifies the number of units of an output unit in the case of a SetQueryItem command.

---

<table>
<thead>
<tr>
<th>NumCopies</th>
<th>Reserved</th>
</tr>
</thead>
</table>

**Figure 87**  
Value Content Format of **NumCopies**

NumCopies : Number of units (pages) of the output.
1 - 65535 units (1 is mandatory)

Vendor unique Value Data (v=1, ValueCode = vendor dependent):
Vendor unique Values concatenated with the company ID shall address a specific vendor unique Value.
6.5. Generic Command

The Generic commands defined in this document are:

- **Send Command**
- **Cancel Command**
- **GetStatus Command**
- **SendStatus Command**

Generic commands are designed to use the packet structure shown below, where the parameters are encapsulated within the command/response packet format. Commands, responses, and their parameters, all have a common format. All commands are defined with a corresponding response, unless otherwise noted.

Some commands may not require parameters, which is dependent on the commands.

---

![Figure 88](image.png)  
**Figure 88** Generic Command and Command Parameters

![Figure 89](image.png)  
**Figure 89** Generic Command Responses and Response Parameters
6.5.1. Implementation Level in Generic Command

DPP devices shall implement DPP defined commands as explained in the following table. Whether the device is a command issuing device or command responding device depends on the meaning of the command and whether the device is an image source device or target device.

<table>
<thead>
<tr>
<th>Table 22</th>
<th>Implementation level in Generic Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command issuing device:</td>
<td>May select and use the necessary command out of the DPP defined commands.</td>
</tr>
<tr>
<td>Command responding device:</td>
<td>Shall be able to understand and execute all the DPP defined commands specified in this document.</td>
</tr>
</tbody>
</table>
6.5.2. Generic Command/Response Packet Format

All commands and responses share a common packet format, as shown below.

<table>
<thead>
<tr>
<th>F</th>
<th>OPT</th>
<th>V</th>
<th>D</th>
<th>L</th>
<th>CmdCode</th>
<th>Count</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Length32 (present when F=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Command Packet Contents</td>
</tr>
</tbody>
</table>

Figure 90 Generic Command/Response Packet Format

F: Length Format

0b = Short format

The valid packet length for this command/response packet will be shown in the 8 bit Length field. The Length32 field shall not be present in this case.

1b = Long format

The valid packet length for this command/response packet will be shown in the 32 bit Length32 field. The Length32 field shall be present in this case.

OPT: Optional Attributes

The meaning of this field is different between commands and responses. In the case of the generic command (D=0), this field shall be filled with 0.

In the case of a response (D=1), this field acts as a Status field.

Status: Status response

0h = No error

1h = Execution Error

2h = Command not supported

V: Vendor Unique Flag

DPP defined commands shall set this field to 0.

0b = DPP defined Commands

1b = Vendor Unique Commands
D: Direction

0b = Command packet
1b = Response packet

L: Level

0b = Command/Response Level
Commands and Responses shall set this field to 0.

CmdCode: Command code
Values are dependent on each command.

Count: Content count
The number of command packet contents following this field.

Length: Packet Length
The length of this Command/Response packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 255 bytes. Command and Response packets with a packet length exceeding 255 bytes shall set the F field to 1, and use the Length32 field.

Length32: Packet Length32
The length of this Command/Response packet structure beyond this field, in bytes. The maximum packet length allowed when using this field is 4,294,967,295 bytes. When this field is used, the value of the Length field is invalid. Command and Response packets with a packet length less than 255 bytes shall set the F field to 0, and use the Length field.

Command Packet Contents: Contents
Command and Response parameters shall be encapsulated in this field.
6.5.3. Generic Command/Response Parameter Format

All generic command parameters and response parameters share a common packet format shown below. The presence and content definition of command and response parameters are dependent on the command.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>OPT</td>
<td>V</td>
<td>D</td>
<td>L</td>
<td>ParaCode</td>
<td>Count</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Length32 (present when F=1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Parameter Contents</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 91** Generic Command/Response Parameter Format

F: Length Format

0b = Short format
The valid packet length for this Parameter packet will be shown in the 8 bit Length field. The Length32 field shall not be present in this case.

1b = Long format
The valid packet length for this Parameter packet will be shown in the 32 bit Length32 field. The Length32 field shall be present in this case.

OPT: Optional Attributes

The meaning of this field is different between commands and responses. In the case of the generic command (D=0), this field shall be filled with 0.
In the case of a response (D=1), this field acts as a Status field.
Status: Status response

0h = No error
2h = Parameter not supported

V: Vendor Unique Flag

Parameters defined within DPP defined commands shall set this field to 0.

0b = DPP defined parameters
D: Direction
   0b = Command parameter
   1b = Response parameter

L: Level
   1b = Parameter Level
   Command and Response parameters shall set this field to 1.

ParaCode: Parameter code
   Values are dependent on each command.

Count: Content count
   The number of parameter contents following this field.

Length: Packet Length
   The length of this Parameter packet structure beyond this field, in bytes. The
   maximum packet length allowed when using this field is 255 bytes. Parameter
   packets with a packet length exceeding 255 bytes shall set the F field to 1, and
   use the Length32 field.

Length32: Packet Length32
   The length of this Parameter packet structure beyond this field, in bytes. The
   maximum packet length allowed when using this field is 4,294,967,295 bytes.
   When this field is used, the value of the Length field is invalid. Parameter
   packets with a packet length less than 255 bytes shall set the F field to 0, and use
   the Length field.

Parameter Contents: Contents
   Contents are dependent on commands
6.5.4. Generic Command Details

There are 4 commands that comprise the Generic command set. This section will describe details of each of these commands.

**Cmd Code:**
Defines the CmdCode field of the command/response packet.

**Function:**
Describes the functionality of the command.

**Command Parameters:**
Defines the Command Packet Contents field of the command packet.

**Return Parameters:**
Defines the Command Packet Contents field of the response packet.

**Command/Response Sequence:**
Describes the command/response transaction.
6.5.4.1. Send

Cmd Code:

03h

Function:
This command will execute the transmission of data. Command parameters with a Para Code of 1h will represent image data. A Send command will execute the transmission of one application image data (See Annex C) within the total DPP image data transfer sequence. A Send command with an image data length of 0 will terminate the total image data transfer sequence. Image source devices shall issue this command in this case. Refer to section 6.7 for a detailed explanation of the total image data transfer sequence.

Image transfer(s) to a target device for one output unit is assumed complete when the image transfer command Send is issued an equal number of times as the Value set for NumPics, or a Send command with an image data length of 0. For example, when the target device supports a NumPics Value of 9 (9 images (frames) per output unit), and the Value of this Item is set to 9 using the SetQueryItem command, the target device will either expect a Send command 9 times, or a Send command with an image data length of 0, as a completion of the image transfer for the output.

Command Parameters:

Para Code: 1h (Image data)

Image data. Refer to ANNEX section for data formats of image data.

Response Parameters:

None

Command/Response Sequence:

CMD: Send
PARA: ImageData

RESP: Send Response
PARA:
6.5.4.2. **Cancel**

**Cmd Code:**

04h

**Function:**

This command will cancel the execution of the previous command issued. Both image source devices and target device may issue this command. This command does not have a corresponding response.

For **Cancel** command, the command issuing device should use the higher priority transfer service than normal application data transfer service, if provided by the lower layer.

In the case of using the Thin Protocol as the lower layer, the command issuing device shall use the Abort service provided as the Thin Session service (See 5.9.5).

**Command Parameters:**

None

**Response Parameters:**

None. *(This command does not have a corresponding response.)*

**Command/Response Sequence:**

CMD: **Cancel**

PARA: -

No Response
6.5.4.3. *GetStatus*

**Cmd Code:**

05h

**Function:**

This command will request the status of the target device. Image source devices may issue this command.

Depending on implementation, there will be cases where the response issuing for the *GetStatus* command will be delayed until a node has finished transferring data.

**Command Parameters:**

None

**Response Parameters:**

Para Code: 1h (target device status)

Response Parameter Content

The format of the Parameter Content field of the Response Parameter is shown below.

<table>
<thead>
<tr>
<th>E</th>
<th>L</th>
<th>F</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 92 Response Parameter Format of GetStatus Command**

- **E**: Output media empty
  - 0b = No error
  - 1b = Empty
- **L**: On-line/Off-line
  - 0b = On line
  - 1b = Off line
- **F**: Fatal error
  - 0b = No error
  - 1b = Error
- **R**: Recoverable error
  - 0b = No error
  - 1b = Error

**Command/Response Sequence:**

CMD: *GetStatus*

PARA: -

RESP: *GetStatus* Response

PARA: Response Parameter
6.5.4.4. **SendStatus**

**Cmd Code:**

06h

**Function:**

This command will provide the status of the device issuing this command. Image target devices may issue this command. This command does not have a corresponding response.

**Command Parameters:**

Para Code: 1h (target device status)

Command Parameter Content

The format of the Command Content field of the Command Parameter is shown below.

![Figure 93 Command Parameter of SendStatus Command](image)

- **E**: Output media empty
  - 0b = No error
  - 1b = Empty
- **L**: On-line/Off-line
  - 0b = On line
  - 1b = Off line
- **F**: Fatal error
  - 0b = No error
  - 1b = Error
- **R**: Recoverable error
  - 0b = No error
  - 1b = Error

**Response Parameters:**

None. (**This command does not have a corresponding response.**)

**Command/Response Sequence:**

CMD: *SendStatus*

PARA: Command Parameter

No Response
6.6. **Vendor Unique Command**

Vendor unique commands shall use the packet structure shown below, where the parameters are encapsulated within the command/response packet format. Command and response definition, and their parameters, are vendor unique.

### 6.6.1. **Vendor Unique Command/Response Packet Format**

All commands and responses share a common packet format, as shown below.

![Vendor Unique Command/Response Packet Format](image)

**V**: Vendor Unique Flag

Vendor unique commands shall set this field to 1.

- 0b = DPP defined Commands
- 1b = Vendor Unique Commands

**CompanyID**: Company ID

This 24 bit company ID value provides the company ID of the vendor defining the particular vendor unique command/response. The company ID value is uniquely assigned to each vendor by the IEEE.

**CompAttrib**: Company Attribute

The definition of this 8 bit value is dependent to the vendor represented by the company ID field.

Usage of all other fields shall be identical to DPP defined Generic commands.
6.7. Command Sequence

6.7.1. Command Sequence Overview

Figure 95 shows the basic command sequence using this command set. The basic flow is performed in the following order.

1) Negotiation (*GetQueryItem* command + *SetQueryItem* command)
2) Data Transfer (*Send* command)

The command issuing device that waits the response for the previous command it issued shall not issue any commands other than the *Cancel* command, *SendStatus* command and *GetStatus* command. *GetStatus* command shall not be sent if the command issuing device is waiting another *GetStatus* response.

Other commands such as the *Cancel* command, *GetStatus* command, and *SendStatus* command, that are not used for (1) negotiation nor (2) data transfer, may be issued at any time if necessary by the image source device and/or target device (depending on the command).
6.7.2. Negotiation

6.7.2.1. Commands for Negotiation

The negotiation process is composed of the following commands:

*GetQueryItem Command*

Image source device may obtain the possible Value(s) of each Item of the target device using the `GetQueryItem` command.

*SetQueryItem Command*

Image source device may select the most suitable Value for each Item and set the target device with this Value using the `SetQueryItem` command.

Negotiation order of the Items is not defined. The image source device may query and set Items in the order according to the priority dependent on the individual image source device.

The `GetQueryItem` command and/or `SetQueryItem` command are not required for those Items using a default Value as shown in Figure 96.

![Diagram showing negotiation process]

**Figure 96** Omission of `GetQueryItem` and/or `SetQueryItem` Command
6.7.2.2. Negotiation Sequence

There are three types of negotiation methods shown below:

1. Sequential Query and Set
2. Multiple Query and Set
3. Negotiation-less

6.7.2.2.1. Sequential Query and Set

It is possible to query and set each Item one by one. Figure 97 shows the negotiation sequence of this case. The GetQueryItem command and SetQueryItem command are issued in pair(s) for each Item. The image source device may issue a GetQueryItem command for one Item to obtain the possible Values for the Item. Then, the image source device may select the most suitable Value for this Item and may issue a SetQueryItem command accompanied with the selected Value for this Item. Other Items in the target device will be set in a sequential manner.

![Diagram of Sequential Query and Set](image-source-device-diagram.png)
6.7.2.2.2. Multiple Query and Set

It is possible to query multiple Items before setting multiple Items. Figure 98 shows the negotiation sequence of this case.

The image source device may issue a *GetQueryItem* command for multiple Items to obtain the possible Values for each Item. Then, the image source device may select the most suitable Value for each Item and may issue a *SetQueryItem* command accompanied with the selected Values for multiple Items.

In the case of setting Items that are dependent upon each other, for instance *OutputOrientation* and *ImageSize*, using multiple query and set, the possible range of Values of one Item may be narrowed after another dependent Item is set.
6.7.2.2.3. Negotiation-less

In the case of using default Values for all Items, no negotiation command is required as shown in Figure 99.

![Diagram showing Negotiation-less](image)

Figure 99 Negotiation-less
6.7.3. Data Transfer

The Data Transfer process is composed of the *Send* command. The *Send* command plays the following two roles according to the included image data length (see section 6.5.4.1.).

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
<th>Image Data Length(L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) <em>Send</em> (Image)</td>
<td>Transmission of one image data within the total DPP image transfer sequence</td>
<td>L &gt; 0</td>
</tr>
<tr>
<td>(2) <em>Send</em> (Termination)</td>
<td>Termination of the total DPP image transfer sequence</td>
<td>L = 0</td>
</tr>
</tbody>
</table>

6.7.3.1. One Picture Per One Output Unit

When the *NumPics* Item is set to 1 (default Value), the image source device may assume that the image transfer for one output unit completes after every image transfer and may or may not issue the *Send* (Termination) command.

If a target device is a printer, it may start printing automatically after one image data transfer completes. Figure 100 shows that the data transfer sequence in the case of one picture (one Application image data) per one output unit (page).
6.7.3.2. Multiple Pictures Per One Output Unit

When the NumPics Item is set to N (N > 1), image transfers for each output unit complete in the following cases:

The number of images transferred from the image source device is equal to a multiple of the NumPics Value.
The Send (Termination) command is issued by the image source device.

If a target device is a printer, it may start printing automatically in either of the two cases above.

Figure 101 shows that the data transfer sequence in the case of multiple pictures (multiple Application image data) per one output unit (page).

![Figure 101 Multiple pictures per one output unit](image)

Where:
- N: Value data of NumPics Item
- m: A natural number
- Response of each command is omitted.
6.7.4. Others

6.7.4.1. Cancel command
The Cancel command may be issued by the image source device and/or target device at any time if necessary (see section 6.5.4.2).

6.7.4.2. GetStatus command
The GetStatus command may be issued by the image source device at any time if necessary (see section 6.5.4.3).

6.7.4.3. SendStatus command
The SendStatus command may be issued by the target device at any time if necessary (see section 6.5.4.4).
7. Configuration ROM

All nodes that implement either the THIN PROTOCOL, or the DIRECT PRINT APPLICATION COMMAND SET of the DIRECT PRINT PROTOCOL shall implement general format configuration ROM in accordance with ISO/IEC 13213:1994, p1212r (under work), IEEE Std 1394-1995 and this standard. General format configuration ROM is a self-descriptive structure defined by IEEE1212. The bus information block and root directory are at fixed locations; all other directories and leaves are addressed by entries in their parent directory.

The figure above shows the areas of the general ROM format described in this document.

All nodes that implement the “THIN PROTOCOL” shall implement the directories described section 7.1.
All nodes shall implement one THIN PROTOCOL unit directory for each command set.

All nodes that implement the “DIRECT PRINT APPLICATION COMMAND SET” shall implement the entries described in section 7.2.
7.1. **Configuration ROM Definition for THIN PROTOCOL**

All nodes that implement the THIN PROTOCOL shall implement general format configuration ROM in accordance with ISO/IEC 13213:1994, IEEE1212r, IEEE Std 1394-1995 and this standard.

7.1.1. **Power reset initialization**

During the initialization process that follows a power reset a device may not be able to respond to Serial Bus request subactions addressed to parts of configuration ROM. When the device has insufficient information to make more than the first quadlet of configuration ROM accessible, it shall return a response value of zero for any read request addressed to FFFF F000 0400 16 or acknowledge the request subaction with ack_tardy, as specified by draft standard IEEE P1394a. Devices shall complete initialization within five seconds of a power reset. Once power reset initialization completes, the device shall make all mandatory configuration ROM entries available. The device should not initiate a Serial Bus reset solely as a consequence of the completion of power reset initialization.

Optional configuration ROM information, such as textual descriptor leaves that identify the device vendor and model, may not be available when power reset initialization completes. The device may add this information to configuration ROM as it becomes available and may initiate a Serial Bus reset to alert other nodes to the changed configuration ROM. The device should initiate a Serial Bus reset if there is no expectation that other nodes would otherwise become aware of changed configuration ROM.
7.1.2. Bus information block

All devices shall implement a bus information block at a base address of FFFF F000 0404_16. For convenience of reference, the format of the bus information block defined by IEEE Std 1394-1995 is shown below. The most recent version of the referenced standard or its supplements shall be referenced.

![Figure 103 Bus information block format](image)

The first quadlet contains the string “1394” in ASCII as the bus_name value required by the CSR Architecture.

The irmc bit (abbreviated as m in the figure above) shall be one if the node is isochronous resource manager capable; otherwise, the irmc value shall be zero.

The cmc bit (abbreviated as c in the figure above) shall be one if the node is cycle master capable; otherwise, this value shall be zero.

The isc bit (abbreviated as i in the figure above) shall be one if the node supports isochronous operations; otherwise, this value shall be zero.

The bmc bit (abbreviated as b in the figure above) shall be one if the node is bus manager capable; otherwise, this value shall be zero.

The cyc_clk_acc field specifies the node’s cycle master clock accuracy in parts per million. If the cmc bit is one, the value in this field shall be between zero and 100. If the cmc bit is zero, this field shall be all ones.
The `max_rec` field defines the maximum data payload size that the node supports. The data payload size applies to block write requests or asynchronous stream packets addressed to the node and to block read responses transmitted by the node. The maximum data payload is equal to $2^{\text{max}_\text{rec}+1}$ bytes.

The `node_vendor_ID` shall be the unique identifier for a company or organization obtained from the IEEE RAC:

The `chip_ID_hi` field concatenated with the `chip_ID_lo` field form a 40-bit chip ID value. The vendor specified by `node_vendor_ID` shall administer the chip ID values. When appended to the `node_vendor_ID` value, these shall form a unique 64-bit value called the EUI-64 (Extended Unique Identifier, 64 bits). These EUI-64 values, are by definition, unique from other EUI-64 identifiers derived from IEEE/RAC-provided company-ID value.
7.1.3. Root directory
Thin Protocol compliant devices shall implement a Configuration ROM with a root directory. The root directory shall contain Module_Vendor_ID and Node_Capabilities entries. The root directory shall also contain at least one Unit_Directory entry defined by this standard.

7.1.3.1. Module_Vendor_ID entry
The Module_Vendor_ID entry is an immediate entry in the root directory that provides the company ID of the vendor that manufactured the module. Figure 104 shows the format of this entry.

```
03 16
```

Module_vendor_ID

Figure 104 Module_Vendor_ID entry format

03_{16} is the concatenation of key_type and key_value for the Module_Vendor_ID entry. The IEEE/RAC uniquely assigns the module_vendor_ID to each module vendor, as specified by ISO/IEC 13213:1994. There is no requirement that the values of module_vendor_ID and node_vendor_ID be equal.

NOTE:
A recommended convention to provide vendor identification in displayable form is to immediately follow the Module_Vendor_ID entry with a textual descriptor leaf entry. This associates an ASCII string with the module vendor. See ISO/IEC 13213:1994 for the specification of textual descriptor leaves; examples are given in Annex D.

7.1.3.2. Node_Capabilities entry
The Node_Capabilities entry is an immediate entry in the root directory that describes node capabilities. Figure 105 shows the format of this entry.

```
0C 16
```

node_capabilities

Figure 105 Node_Capabilities entry format

0C_{16} is the concatenation of key_type and key_value for the Node_Capabilities entry.
The node_capabilities field contains subfields, specified by ISO/IEC 13213:1994. Devices shall implement the SPLIT_TIMEOUT register, the 64-bit fixed addressing scheme, the STATE_CLEAR.lost bit and the STATE_CLEAR.dreq bit and indicate this by setting the spt, 64, fix, lst and drq bits to one. If no other node_capabilities bits are one this results in a value of 0083C016.

7.1.3.3. Unit_Directory entry
The Unit_Directory entry is a directory entry in the root directory that describes the location of a unit directory within configuration ROM. There may be more than one unit directory; each unit directory shall be located by a separate Unit_Directory entry. Figure 106 shows the format of this entry.

![Figure 106: Unit_Directory entry format](image)

D1_{16} is the concatenation of key_type and key_value for the Unit_Directory entry.
The indirect_offset field specifies the number of quadlets from the address of the Unit_Directory entry to the address of the unit directory within configuration ROM.

7.1.4. Unit directory
The root directory shall also contain at least one Unit_Directory entry defined by this standard. The unit directory shall contain Unit_Spec_ID and Unit_SW_Version entries, as specified by ISO/IEC 13213:1994.
The unit directory shall also contain a Command_Set_Spec_ID entry, a Command_Set entry, a Command_Set_Details entry, and a Connection_Register entry, as specified by this standard. The unit directory may also contain a Write_Transaction_Interval entry.
7.1.4.1. Unit_Spec_ID entry
The Unit_Spec_ID entry is an immediate entry in the unit directory that specifies the organization responsible for the architectural definition of the device. Figure 107 shows the format of this entry.

```
<table>
<thead>
<tr>
<th>Most significant</th>
<th>Least significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>12_{16}</td>
<td>00A02D_{16} (1394TA)</td>
</tr>
</tbody>
</table>
```

**Figure 107**  Unit_Spec_ID entry format

12_{16} is the concatenation of key_type and key_value for the Unit_Spec_ID entry.
00A02D_{16} is the unit_spec_ID obtained by 1394TA from the IEEE/RAC. The value indicates that the 1394TA is responsible for the software interface definition.

7.1.4.2. Unit_SW_Version entry
The Unit_SW_Version entry is an immediate entry in the unit directory that, in combination with the unit_spec_id, specifies the software interface of the device. Figure 108 shows the format of this entry.

```
<table>
<thead>
<tr>
<th>Most significant</th>
<th>Least significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>13_{16}</td>
<td>0A6BE2_{16}</td>
</tr>
</tbody>
</table>
```

**Figure 108**  Unit_SW_Version entry format

13_{16} is the concatenation of key_type and key_value for the Unit_SW_Version entry.
0A6BE2_{16} is the unit_sw_version value that indicates that the device conforms to this standard.
7.1.4.3. Command_Set_Spec_ID entry
The Command_Set_Spec_ID entry is an immediate entry that, when present in the unit directory, specifies the organization responsible for the command set definition for the device. Figure 109 shows the format of this entry.

Figure 109  Command_Set_Spec_ID entry format

38_{16} is the concatenation of key_type and key_value for the Command_Set_Spec_ID entry. The command_set_spec_ID is an organizationally unique identifier obtained from the IEEE/RAC. The organization to which this 24-bit identifier has been granted is responsible for the definition of the command set implemented by the device.

7.1.4.4. Command_Set entry
The Command_Set entry is an immediate entry that, when present in the unit directory, in combination with the command_set_spec_ID specifies the command set implemented by the device. Figure 110 shows the format of this entry.

Figure 110  Command_Set entry format

39_{16} is the concatenation of key_type and key_value for the Command_Set entry. The value of command_set shall be specified by the owner of command_set_spec_ID.
7.1.4.5. **Command_Set_Details entry**
The Command_Set_Details entry is an immediate entry that, when present in the unit directory, specifies the details, such as revision level of the command set implemented by the device. Figure 111 shows the format of this entry.

![Command_Set_Details entry format](image1)

3A16 is the concatenation of key_type and key_value for the Command_Set_Revision entry. The value of command_set_revision shall be specified by the owner of command_set_spec_ID and command_set_spec_ID.

7.1.4.6. **Connection_Register entry**
The Connection_Register entry is an immediate entry in the unit directory that specifies the base address of the device’s Connection Register. Figure 112 shows the format of this entry.

![Connection_Register entry format](image2)

7B16 is the concatenation of key_type and key_value for the Connection_Register entry. The csr_offset field shall contain the offset, in quadlets, from the base address of initial register space, FFFF F000 000016, to the base address of the Connection register for the device. All device CSR’s shall be located at or above address FFFF F001 000016; therefore the value of csr_offset shall not be less than 400016.

7.1.4.7. **Write Transaction Interval entry (optional)**
The Write_Transaction_Interval entry is an optional immediate entry in the unit directory that specifies the minimum time required between write transactions to this node to avoid write transaction retries. Nodes which require considerable amount of time to process write transactions should implement this register. Figure 113 shows the format of this entry.

![Write_Transaction_Interval entry format](image3)
3C_{16} is the concatenation of key_type and key_value for the Write Transaction Interval entry. The Write_Transaction_Interval entry shall contain the interval time value in microseconds.
7.2. Configuration ROM Definition for Direct Print Command Set

All nodes that implement the “DIRECT PRINT APPLICATION COMMAND SET” shall implement general format configuration ROM in accordance with ISO/IEC 13213:1994, IEEE1212r, IEEE Std 1394-1995 and this standard.

7.2.1. Command_Set_Spec_ID entry

The Command_Set_Spec_ID entry is an immediate entry that, when present in the unit directory, specifies the organization responsible for the command set definition for the target. Figure 114 shows the format of this entry.

![Figure 114 Command_Set_Spec_ID entry format](image)

38H is the concatenation of key_type and key_value for the Command_Set_Spec_ID entry.
00A02DH is the Command_Set_Spec_ID obtained by 1394TA from the IEEE/RAC. The value indicates that the 1394TA is responsible for this command set definition.

7.2.2. Command_Set entry

The Command_Set entry is an immediate entry that, when present in the unit directory, in combination with the command_set_spec_ID specifies the command set implemented by the target. Figure 115 shows the format of this entry.

![Figure 115 Command_Set entry format](image)

39H is the concatenation of key_type and key_value for the Command_Set entry.
B081F2H is the Command_Set obtained by the DIRECT PRINT APPLICATION COMMAND SET from the 1394TA. The value indicates this command set.
7.2.3. Command_Set_Details entry

The Command_Set_Details entry is an immediate entry that specifies the revision number, and details of the DPP command set implemented. Figure 116 shows the format of this entry.

3A16 is the concatenation of key_type and key_value for the Command_Set_Details entry.

In this field, the 2 least significant bits specify the DPP device type. The least significant bit when 1 will represent an image target device, and the second significant bit when 1 will represent an image source device.

The 2 least significant bytes represent the revision number of the command set. 4 bits will represent a digit in hexadecimal, with the most significant byte representing the 2 digits above the decimal point, and the next significant byte representing 2 decimal places below the decimal point.

For example, version 1.0 will be represented as 01000xh, and version 2.25 will be represented as 02250xh, where x is the least significant byte including the S bit and the T bit.
7.3. Instance Directory

Devices supporting the Thin protocol shall implement the Instance directory currently being discussed in p(IEEE)1212r or equivalent. Following sections describe the entries which comprise the Instance directory structure. Refer to the p1212r documents, or ANNEX D for details.

7.3.1. Root Directory

7.3.1.1. Instance_Directory entry

This entry is used to provide functional information on a instance that the node supports. An Instance_directory entry represents an instance within a node, and points to a lower-level directory (Instance_Directory) that contains one function_class entry describing the function class of an instance within a node.

<table>
<thead>
<tr>
<th>D8h</th>
<th>Instance_Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24</td>
</tr>
</tbody>
</table>

Figure 117 Instance_Directory entry format
7.3.2. Instance Directory

An Instance Directory is a set of information that uniquely represents instances. The directory consists of 1 function_class entry that identifies the class category of the instance, zero or more Unit_Directory entries as pointers to associated unit directories, and zero or more Feature_directory entries as pointers to associated Feature directories. In case a instance consists of “sub” instances that can be represented, an instance directory entry (or an Instance_List entry) can be included to extend the instance hierarchy.

7.3.2.1. Function_Class entry

This entry is used to identify the functional capability of the instance within the node. This 24 bit immediate Function_Class entry value represents the class categorization of the instance. 1 Function_Class entry shall be present for each instance directory. In addition to the one entry representing the instance, additional Function_Class entries can be present to provide alias information. The class categorization and corresponding entry values are specified in p1212r. Refer to the p1212r draft specification for device categorization and corresponding values.

<table>
<thead>
<tr>
<th>19h</th>
<th>Function_Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24</td>
</tr>
</tbody>
</table>

Figure 118 Function_Class entry format

7.3.2.2. Unit_Directory entry

This entry, when used in the Instance directory shall point to a unit directory that is associated with the given instance.

<table>
<thead>
<tr>
<th>D1h</th>
<th>Unit_Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24</td>
</tr>
</tbody>
</table>

Figure 119 Unit_Directory entry format
Annex A. DPP Implementation Examples

Annex A-1..A-5 shows examples in which a digital still (DS) camera and a DPP printer realize the direct printing. The assumptions of these examples are as follows:

- The DS camera and the DPP printer are connected through 1394 bus.
- The DS camera supports Thin Protocol and DPP command set as an image source device.
- Using the configuration ROM (unit directory / instance directory), the DS camera has already detected the DPP printer supporting Thin Protocol and DPP command set as a target device (See Annex E.)

A-1. Total DPP Flow Example

Figure 120 shows an example of a total DPP flow.

(1) The application of the DS camera will establish a connection with the DPP printer using connect request of the Thin Protocol Service (See 5.8.1.).

(2) The application of the DS camera will execute the negotiation and the data transfer to the DPP printer using DPP command set. The detail examples of this step are shown in Annex A-2 and A-3.

(3) The application of the DS camera will close the connection with the DPP printer using disconnect request of the Thin Protocol Service (See 5.8.3.)

Figure 120 Total DPP Flow Example
A-2. An Example of Sequential Query and Set

In the following example, negotiation is performed using sequential query and set, and the target device (printer) starts printing after every application image data transfer completes (NumPics = 1).

Figure 121 An Example of Sequential Query and Set
A-3. An Example of Multiple Query and Set

In the following example, negotiation is performed using multiple query and set, and the target device (printer) starts printing after every four application image data transfers complete (NumPics = 4). In this case, Send (Termination) shall be issued.

![Diagram](image)

---

**Figure 122 An Example of Multiple Query and Set**
A-4. Packet Format Implementation

(1) Issuing GetQueryItem Command

Figure 123 shows an example of the GetQueryItem command used to query multiple Items in one time.

Figure 123 An Example of GetQueryItem Command

(2) Responding to GetQueryItem Command

Figure 124 shows an example of the GetQueryItem response to the GetQueryItem command shown in Figure 123.

In the case of querying all ten Items defined in DPP in one GetQueryItem command (querying multiple items in one time), the total response packet size of the GetQueryItem command will be approximately 50 quadlets (200 bytes).

Figure 124 An Example of GetQueryItem response
(3) Issuing SetQueryItem Command

Figure 125 shows an example of the SetQueryItem command according to the results of the GetQueryItem response shown in Figure 124.

(4) Responding to SetQueryItem Command

a) Succeeding in Setting Items

Figure 126 shows an example of the SetQueryItem response to the SetQueryItem command shown in Figure 125, in case the SetQueryItem has been successful.

b) Failing in Setting Items

Figure 127 shows an example of a SetQueryItem response to the SetQueryItem command shown in Figure 125, in case the SetQueryItem has been unsuccessful.
(5) Issuing *Send* Command

Figure 128 shows an example of the *Send* command.

![Figure 128 An Example of Send Command](image)

(6) Responding to *Send* Command

a) Succeeding in Image Data Transfer

Figure 129 shows an example of the *Send* response to a successful *Send* command shown in Figure 128.

![Figure 129 An Example of Send Response – 1](image)

b) Failing in Image Data Transfer

Figure 130 shows an example of the *Send* response to an unsuccessful *Send* command shown in Figure 128.

![Figure 130 An Example of Send Response - 2](image)
Annex B. Negotiation Implementation

B-1. Steps of Negotiation in Sequential Query and Set

Figure 131 shows an example of negotiation steps using sequential query and set. One step of query and set determines the value of one Item.

![Diagram of negotiation steps](image-url)

---

**Figure 131** Steps of Negotiation in Sequential Query and Set
B-2. Steps of Negotiation in Multiple Query and Set

Figure 132 shows an example of negotiation steps using multiple query and set. One step of query and set determines the values of several Items, but there is a possibility that some Items are not set depending on the Items set before.

Figure 132 Steps of Negotiation in Multiple Query and Set
Annex C. Image data format Details

C-1. Overview

DPP defines the following four image data formats. Categorization of these formats is shown in Figure 133.

Table 24 DPP defined image data formats

<table>
<thead>
<tr>
<th>Image data format</th>
<th>Description</th>
<th>Implementation Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) rawRGB</td>
<td>RGB chunky data</td>
<td>Mandatory</td>
</tr>
<tr>
<td>(2) D-rawRGB</td>
<td>RGB chunky data with dummy channel for quadlet (32bit) allocation</td>
<td>Option</td>
</tr>
<tr>
<td>(3) Exif</td>
<td>Exif-JPEG</td>
<td>Option</td>
</tr>
<tr>
<td>(4) JFIF</td>
<td>JFIF</td>
<td>Option</td>
</tr>
</tbody>
</table>

Figure 133 Categorization of Image Data Format
C-2.  rawRGB

Image Data Arrangement

Figure 134 shows the image data arrangement of the rawRGB image data format representing RGB chunky data. One pixel’s data is composed of R(Red), G(Green) and B(Blue) channels and is arranged in order from the starting pixel to the final pixel of raster scanning.

If the total data size in bytes is NOT multiple of four, fill 00h for quadlet allocation.

<table>
<thead>
<tr>
<th>Ri</th>
<th>Gi</th>
<th>Bi</th>
<th>Rj</th>
<th>Gj</th>
<th>Bj</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2</td>
<td>B2</td>
<td>R3</td>
<td>G3</td>
<td>B3</td>
<td>R4</td>
</tr>
<tr>
<td>B3</td>
<td>R4</td>
<td>G4</td>
<td>B4</td>
<td>R5</td>
<td>G5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where
- Ri: i th Data for red channel (i=1..N)
- Gi: i th Data for green channel (i=1..N)
- Bi: i th Data for blue channel (i=1..N)
- N: Total number of pixels
  (i=1 represents the starting pixel and i=N represents the final pixel of raster scanning)

Figure 134  Image Data Arrangement of rawRGB

Bit Depth of Image Data

All channels are composed of 8 bit depth image data.

Color Space

Image data shall be designed to be seen at the viewing environment of sRGB shown below.

Table 25  Viewing Environment of sRGB

<table>
<thead>
<tr>
<th>Viewing flare</th>
<th>1.0%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image surround</td>
<td>20%</td>
</tr>
<tr>
<td>Illuminance/Luminance level</td>
<td>80cd/cm²</td>
</tr>
<tr>
<td>Adaptive white</td>
<td>x=0.3127, y=0.3290 (CIE D65)</td>
</tr>
</tbody>
</table>
C-3.  **D-rawRGB**  

**Image Data Arrangement**  

Figure 135 shows the image data arrangement of *D-rawRGB* image data format representing RGB chunky data with dummy data for quadlet allocation. One pixel’s data is composed of D(Dummy), R(Red), G(Green) and B(Blue) channels and is arranged in order from the starting pixel to the final pixel of raster scanning.

<table>
<thead>
<tr>
<th>00h(Dummy)</th>
<th>R1</th>
<th>G1</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h(Dummy)</td>
<td>R2</td>
<td>G2</td>
<td>B2</td>
</tr>
<tr>
<td>00h(Dummy)</td>
<td>R3</td>
<td>G3</td>
<td>B3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00h(Dummy)</td>
<td>BN</td>
<td>GN</td>
<td>BN</td>
</tr>
</tbody>
</table>

Where  
- Ri: *i* th Data for red channel (i=1..N)  
- Gi: *i* th Data for green channel (i=1..N)  
- Bi: *i* th Data for blue channel (i=1..N)  
- N: Total number of pixels  

(i=1 represents the starting pixel and i=N represents the final pixel of raster scanning)

**Figure 135  Image Data Arrangement of D-rawRGB**

**Bit Depth of Image Data**  

All channels including the dummy channel are composed of 8 bit depth image data.

**Color Space**  

Image data shall be designed to be seen at the viewing environment of sRGB shown in Table 25.
C-4. Exif

Image File Format

Image file format shall conform to Exif Ver.2. See Exif Ver.2 Specification for details.

Additional Specification

Table 26 shows additional restrictions to the choices defined in Exif Ver.2 specification. Target device supporting Exif as image data format shall be capable of handling Exif image files with any variations shown in this figure.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Additional Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Format</td>
<td>JPEG Compression</td>
</tr>
<tr>
<td>Pixel Sampling</td>
<td>YCC 4:2:0 or 4:2:2</td>
</tr>
<tr>
<td>Restart Marker</td>
<td>May be inserted</td>
</tr>
<tr>
<td>Image Width, Image Length</td>
<td>Shall conform to the negotiated value of ImageSize</td>
</tr>
<tr>
<td>Color Space</td>
<td>sRGB</td>
</tr>
<tr>
<td>Byte Order</td>
<td>Little Endian or Big Endian</td>
</tr>
</tbody>
</table>

Exif Tag Handling

Several tags defined in the Exif specification need the handling guidelines shown in Table 27.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Exif Tag handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orientation Tag</td>
<td>Target device shall use the negotiated value of the DPP defined Image Orientation Item instead of the Exif defined Orientation tag included in the Exif image file.</td>
</tr>
<tr>
<td>Xresolution Tag, Yresolution Tag, ResolutionUnit Tag</td>
<td>Target device may ignore the Xresolution, Yresolution and ResolutionUnit tags included in the Exif image file.</td>
</tr>
</tbody>
</table>
**Image Data Arrangement**

Figure 136 shows the image data arrangement of transmitting an Exif image file.

If the total data size in bytes of the Exif image is NOT a multiple of four, fill 00h for quadlet allocation.

<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where

- \( D_i \): \( i \)th data element of Exif image file \((i=1..N)\)
- \( N \): Total data size of Exif image file (bytes)

**Figure 136**  Image Data Arrangement of *Exif*
C-5. **JFIF**

**Image File Format**

Image File Format shall conform to JFIF v1.02. See JFIF v1.02 Specification for details.

**Additional Specification**

Table 28 shows additional restrictions to the choices defined in JFIF v1.02 Specification. Target device supporting JFIF as image data format shall be capable of handling JFIF image files with any variations shown in this figure.

**Table 28 Additional Restrictions**

<table>
<thead>
<tr>
<th>Data Format</th>
<th>JPEG compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel Sampling</td>
<td>YCC 4:2:2 or 4:2:0</td>
</tr>
<tr>
<td>DHT marker</td>
<td>up to 4</td>
</tr>
<tr>
<td>DQT marker</td>
<td>up to 3</td>
</tr>
<tr>
<td>DQT tables</td>
<td>1 table per DQT or single DQT defines all tables</td>
</tr>
<tr>
<td>DHT tables</td>
<td>1 table per DHT or single DHT defines all tables</td>
</tr>
<tr>
<td>X,Y density in APP0</td>
<td>recommended to reflect X and Y density in printing if units for X and Y density is defined.</td>
</tr>
<tr>
<td>Color Space</td>
<td>recommended use of sRGB in printing</td>
</tr>
</tbody>
</table>

**Image Data Arrangement**

Figure 137 shows the image data arrangement of transmitting JFIF image file. If the total data size in byte of JFIF image is NOT multiple of four, fill 00h for quadlet allocation.

```
\[\begin{array}{cccc}
D_1 & D_2 & D_3 & D_4 \\
D_5 & D_6 & D_7 & D_8 \\
\cdots \cdots \\
D_{N-1} & D_N & 00h & 00h \\
\end{array}\]
```

Where

- $D_i$: $i$ th data of JFIF image file ($i=1..N$)
- $N$: Total data size of JFIF image file(bytes)

**Figure 137 Image Data Arrangement of JFIF**
Annex D. Sample Configuration ROM

Configuration ROM is located at a base address of FFFF F000 040016 within a node’s initial memory space. The requirements for general format configuration ROM for devices are specified in section 7. This annex contains illustrations of typical configuration ROM for simple DPP devices.

D-1 Basic device Sample

Figure 138 below shows the bus information block, root directory, unit directory and function directory for a basic thin protocol device.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>ROM CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0416</td>
<td>3133393416 (ASCII“1394”)</td>
</tr>
<tr>
<td>Node options(00FF2000)16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node vendor ID</td>
<td>Chip_id_hi</td>
<td></td>
</tr>
<tr>
<td>Chip_id_lo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Directory length</td>
<td>Root directory CRC</td>
<td></td>
</tr>
<tr>
<td>0316</td>
<td>Module_vendor_id</td>
<td></td>
</tr>
<tr>
<td>8116</td>
<td>Text leaf offset</td>
<td></td>
</tr>
<tr>
<td>0C16</td>
<td>Node capabilities (0083C016)</td>
<td></td>
</tr>
<tr>
<td>D816</td>
<td>Instance Directory offset(2)</td>
<td></td>
</tr>
<tr>
<td>D116</td>
<td>Unit Directory offset(4)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Instance Directory CRC</td>
<td></td>
</tr>
<tr>
<td>1916</td>
<td>Function_Class</td>
<td></td>
</tr>
<tr>
<td>D116</td>
<td>Unit Directory offset(1)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unit directory CRC</td>
<td></td>
</tr>
<tr>
<td>1216</td>
<td>Unit_Spec ID (00A02D16)</td>
<td></td>
</tr>
<tr>
<td>1316</td>
<td>Unit_sw_version (0A6BE216)</td>
<td></td>
</tr>
<tr>
<td>3816</td>
<td>Command_set_spec_id (00A02D16)</td>
<td></td>
</tr>
<tr>
<td>3916</td>
<td>Command_set (B081F216)</td>
<td></td>
</tr>
<tr>
<td>3A16</td>
<td>Command_set_details</td>
<td></td>
</tr>
<tr>
<td>7B16</td>
<td>Connection register offset(00400016)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 138 Configuration ROM Sample of Basic Device
Root directory
The node_options field represents a collection of bits and fields specified in section 7.1.2. The value shown, 00FF 200016, represents basic characteristics of a device that is not isochronous capable. This value is composed of a cyc_clk_acc field with a value of FF16 and a max_rec value of two. The max_rec field encodes a maximum payload of eight bytes in block write requests addressed to the device.

The Node_Capabilities entry in the root directory, with a key field of 0C16, has a value where the spt, 64, fix, 1st and drq bits are all one. This is a minimum requirement for devices.

The Module_Vendor_ID entry in the root directory, with a key field of 0316, is immediately followed by a textual descriptor leaf entry, with a key field of 8116, whose indirect_offset value points to a leaf that contains an ASCII string that identifies the vendor. Although this textual descriptor leaf is not shown, if it were placed immediately after the 23 quadlets illustrated, the value of indirect_offset would be 13. See the IEEE1212 specification for an example of a text leaf.

The Unit_Directory entry in the root directory, with a key field of D116, has an indirect_offset value of one that points to the unit directory.

The Instance_Directory entry in the root directory, with a key field of D816, represents a functional instance of the node, and has an indirect_offset value of one that points to the Instance directory.

Unit directory
The Command_Set_Spec_ID and Command_Set entries, with a key field of 3816 and 3916, respectively, are expected to define the command set used by the device.

The connection_register entry in the unit directory, with a key field of 7B16, has a csr_offset value of 00 400016 in this example, that indicates that the Connection register has a base address of FFFF F001 000016 within the node’s initial memory space.

Instance directory
The Function_Class entry, with the key field value of 1916 represent a function_class within the node.

The following unit directory offset entry will point to the associated unit directory for this instance.
**D-2. Multi-function device sample**

Figure 139 below shows the bus information block, root directory, unit directory and Instance directory for a thin protocol device with multiple (2) instances with different functionality, but instances sharing the same thin protocol.

<table>
<thead>
<tr>
<th>4</th>
<th>04_{16}</th>
<th>ROM CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3133934_{16} (ASCII“1394”)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Node options(00FF2000)_{16}</td>
<td></td>
</tr>
<tr>
<td>Node vendor ID</td>
<td>Chip_id_hi</td>
<td></td>
</tr>
<tr>
<td>-direction length</td>
<td>Root directory CRC</td>
<td></td>
</tr>
<tr>
<td>03_{16}</td>
<td>Module_vendor_id</td>
<td></td>
</tr>
<tr>
<td>81_{16}</td>
<td>Text leaf offset</td>
<td></td>
</tr>
<tr>
<td>0C_{16}</td>
<td>Node capabilities (0083C0_{16})</td>
<td></td>
</tr>
<tr>
<td>D8_{16}</td>
<td>Instance Directory offset(3)</td>
<td></td>
</tr>
<tr>
<td>D8_{16}</td>
<td>Instance Directory offset(5)</td>
<td></td>
</tr>
<tr>
<td>D1_{16}</td>
<td>Unit Directory offset(7)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Instance Directory CRC</td>
<td></td>
</tr>
<tr>
<td>19_{16}</td>
<td>Function_Class</td>
<td></td>
</tr>
<tr>
<td>D1_{16}</td>
<td>Unit Directory offset(4)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Instance Directory CRC</td>
<td></td>
</tr>
<tr>
<td>19_{16}</td>
<td>Function_Class</td>
<td></td>
</tr>
<tr>
<td>D1_{16}</td>
<td>Unit Directory offset(1)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Unit directory CRC</td>
<td></td>
</tr>
<tr>
<td>12_{16}</td>
<td>Unit_Spec ID (00A02D_{16})</td>
<td></td>
</tr>
<tr>
<td>13_{16}</td>
<td>Unit_sw_version (0A6BE2_{16})</td>
<td></td>
</tr>
<tr>
<td>38_{16}</td>
<td>Command_set_spec_id (00A02D_{16})</td>
<td></td>
</tr>
<tr>
<td>39_{16}</td>
<td>Command_set (B081F2_{16})</td>
<td></td>
</tr>
<tr>
<td>3A_{16}</td>
<td>Command_set_details</td>
<td></td>
</tr>
<tr>
<td>7B_{16}</td>
<td>Connection register offset(004000_{16})</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 139** Configuration ROM Sample of Multi-function device
**Instance directory**
There are 2 Instance directory entries present, 1 for each functional instance.

**Instance directory**
2 Instance directories are present for each functionality. Each Instance directory has a function_class entry. Each Instance directory has a unit directory offset entry.
Though the unit directory offset entries have different offset values of 4 and 1 respectively, the both reference the same unit directory.
D-3. Multi-Protocol device sample

Figure 140 below shows the unit directory and Instance directory for a single function device compliant with the thin protocol as well as another protocol.

<table>
<thead>
<tr>
<th>2</th>
<th>Function Directory CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>19_{16}</td>
<td>Function_Class</td>
</tr>
<tr>
<td>D1_{16}</td>
<td>Unit Directory offset(2)</td>
</tr>
<tr>
<td>D1_{16}</td>
<td>Unit Directory offset(8)</td>
</tr>
<tr>
<td>6</td>
<td>Unit directory CRC</td>
</tr>
<tr>
<td>12_{16}</td>
<td>Unit_Spec_ID (00A02D_{16})</td>
</tr>
<tr>
<td>13_{16}</td>
<td>Unit_sw_version (0A6BE2_{16})</td>
</tr>
<tr>
<td>38_{16}</td>
<td>Command_set_spec_id (00A02D_{16})</td>
</tr>
<tr>
<td>39_{16}</td>
<td>Command_set (B081F2_{16})</td>
</tr>
<tr>
<td>3A_{16}</td>
<td>Command_set_details</td>
</tr>
<tr>
<td>7B_{16}</td>
<td>Connection register offset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unit directory length</th>
<th>Unit directory CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>12_{16}</td>
<td>Unit_Spec_ID</td>
</tr>
<tr>
<td>13_{16}</td>
<td>Unit_sw_version</td>
</tr>
</tbody>
</table>

Contents dependent on protocol

Figure 140  Configuration ROM Sample of Multi-Protocol device

Instance directory

The instance directory has a function_class entry representing the functionality of the instance within the node.

The 2 unit directory offset entries represent pointers to the 2 unit directories that this instance is associated with, the offset entry with the value of 2 representing the thin protocol unit directory, and the entry with the value of 8 representing the other unit directory respectively.
Annex E. Enumeration of DPP devices using Configuration ROM

Compliance to the DPP configuration ROM requirements will enable applications easy and efficient discovery of DPP devices, especially in an environment where multiple devices of various functionality and protocol support are connected together. This annex will give an example of a DPP device enumeration scheme to explain the utilization of the configuration ROM.

The figure below illustrates an example of a multi-node IEEE1394 topology of devices with various functionality and protocol support.

![Multiple Device Topology](image)

The following devices are connected together in the topology above:

- Device A, which is a VCR supporting the AVC/FCP protocol
- Device B, which is a Digital Still Camera (DSC) supporting the DPP Application command/Thin protocol
- Device C, which is a printer supporting the SBP-2 protocol
- Device D, which is a printer supporting the DPP Application command/Thin protocol
- Device E, which is a disk device supporting the DPP Application command/Thin protocol

All of the 1394 devices above are required to implement the configuration ROM. In addition, the protocols that are supported by the devices above each have separate value definitions of entries in the unit directory (unit_spec_id, unit_sw_version) that distinguish each protocol.

In this example, it is assumed that all 1394 devices implement the instance directory to contain functionality information of the device. (DPP devices are required to implement the instance directory.)

The following explanation will describe an example of an efficient device discovery scheme where Device B, the DPP DSC will search for the appropriate device to execute a direct-print task and
achieve an image output from a printer.
Device B will parse the Instance directory of the configuration ROM of each connected device to retrieve device functional information, by using basic 1394 read transactions.
Device B will first search the Instance directory to look for a “printer” instance. The figure below will show that the information in the instance directory describe Device C and Device E as “printer” devices.

![Diagram](image)

**Figure 142**  **Step1: Printer Device Discovery**

Device B has succeeded in sorting out printing devices from other devices which do not match the needs for the task. However, there is no information at this stage in the discovery scheme whether Device C or Device D support the DPP protocol that Device B supports.
Device B will then parse the unit directories of Device C and Device D to see if either of the 2 devices support the DPP Application command / Thin Protocol. In this case, Device D is the device with a unit directory representing the DPP protocol, whereas Device C supports a different protocol.

![Diagram](image)

**Figure 143**  **Step2: DPP Printer Device Discovery**
Device B has now successfully discovered the appropriate device to achieve an image output, and supports the DPP protocol. Device B will further parse the DPP unit directory of Device D to get the entry information (Connection register address) for the DPP protocol. Device B will utilize the DPP protocol to make a connection to Device D.

The figure below show the 2 types of device information presented in the configuration ROM (i.e. ROM level device discovery) for each device. Other information contained in the configuration ROM (such as bus information, node capabilities, vendor information) are omitted to make the explanation simple.

![Configuration ROM Information for Device Discovery](image)

**Figure 144** Configuration ROM Information for Device Discovery

Though the example given describes the discovery scheme by parsing the instance directories to sort the device functionality first, there are no rules as to the order in which the ROM should be parsed. If Device B were to parse the unit directories first and initially sort out devices supporting the DPP protocols, it would first distinguish Device D and Device E, instead of Device C and Device D.

In either case, implementing and parsing both the DPP unit directory and the p1212r Instance directory will enable an efficient and high level of device enumeration at ROM level.
Annex F. Thin Protocol State Machine (Informative)

Annex F contains the state machines and the state machine notes for the Thin Session and the Thin Transaction. Note that the descriptions in this annex are informative.

F-1. Thin Session Connect State Machine

Figure 145 Thin Session connect state machine
**State C0: Connect Ready.** This is the entry point of the Thin Session connect state machine. A node is ready to send or receive a connect request. All timers are stopped at the entry of this state.

**State C1: Connect Request.** When a connect request is issued from an application, a node sends a connect request. In this state, a node is waiting to receive a connect response from the responder node.

**State C2: Connect Response.** When a connect request is received, a node indicates a connect request to an application. In this state, a node is waiting to receive a response from an application.

**State C3: Connected.** At this point, a connection between the initiator node and responder node is established. A node stays in this state until an internal or external event.

**State C4: Reconnect Start.** When a bus reset or an unexpected error occurred, a reconnect process is started. A node sends a reconnect request.

**State C5: Reconnect Response.** A reconnect request is received from the other node. Even if a reconnect request was sent, a node sends a reconnect response.
F-2. Thin Session Outbound Data Transfer State Machine

**S0: Send Data Ready**

- **initialize**
  - **All:** S0

- **S0:**
  - **S0:** S1a
    - session request of command
  - **S0:** S1b
    - session indication of command ID
    - transaction request of SDU send with single data or first segment data
  - **S0:** S1c
    - session response of command
    - transaction request of SDU send with single data or first segment data
  - **S0:**
    - S1:S1
      - transaction confirmation with SDU ready and interim / last segment data transfer next
    - S1:S0a
      - transaction request of SDU send with segment data
  - **S0:**
    - S2:S1
      - internal event of reconnect complete
      - transaction request of SDU send with previous data
  - **S0:**
    - S2:S2
      - session request of abort

- **S1:** Send Data

- **S2:** Reconnect Wait

- **S3:** Data Transfer Stop for Abort Request

- **S4:** Abort Request

- **S5:** Data Transfer Stop

**Figure 147** Thin Session outbound data transfer state machine
Note that this state machine is intended for a single command data transfer. To handle a multiple command data transfer, a node may have an individual state machine for each command, e.g.

**State S0: Send Data Ready.** This is the entry point of the Thin Session outbound state machine. An outbound data transfer process is initialized and ready for sending data.

**State S1: Send Data.** A command request or response is issued from an application. In case of a command request, an indication of applied command ID is indicated to an application. A node stays in this state until the application data is sent completely.

**State S2: Reconnect Wait.** If a bus reset or an unexpected error is indicated, a node waits to receive an internal event of reconnect complete. When reconnect sequence is completed, the previous data is sent again.

**State S3: Data Transfer Stop for Abort Request.** An abort request is issued from an application during a data transfer. A node sends a data transfer stop request and waits to receive a transaction confirmation.

**State S4: Abort Request.** An abort request including a higher layer abort command is sent. A node is waiting to receive a transaction confirmation.

**State S5: Data Transfer Stop.** An internal event of transfer abort is occurred during a data transfer. A node sends a data transfer stop request and waits to receive a transaction confirmation.
F-3. Thin Session Inbound Data Transfer State Machine

R0: Receive Data Ready
- initialize
- R0:R1
  - transaction indication of SDU send
  - single data/last segment data and SDU register becomes available
  - transaction response with SDU ready
  - session indication or confirmation of command
  - abort command and SDU register becomes available
  - internal event of transfer abort
  - transaction response with SDU ready
  - SDU can not accept and SDU register becomes available
  - transaction response with SDU reject
  - session indication of error
- All:R0

R1: Receive Data
- transaction indication of SDU send
- R1:R0a
  - session indication or confirmation of command
- R1:R0b
  - transaction response with SDU ready
- R1:R0c
  - transaction response with SDU reject
- All:R3

R2: Wait Next Data
- R2:R1
  - transaction indication of SDU send

R3: Receive Transfer Stop
- R3:R0
  - transaction indication of SDU stop

Figure 148  Thin Session inbound data transfer state machine

Note that this state machine is intended for a single command data transfer. To handle a multiple command data transfer, a node may have an individual state machine for each command, e.g.

State R0: Receive Data Ready. This is the entry point of the Thin Session inbound state machine. An inbound data transfer process is initialized and ready for receiving data.

State R1: Receive Data. An application data is received. A node waits for an SDU register to be ready, and sends a response.

State R2: Wait Next Data. When a node receives a first or interim segment data, the node waits for the next segment data in this state.

State R3: Receive Transfer Stop. A data transfer stop request is received during a data transfer. A node waits for an SDU register to be ready, and sends a response.
F-3. Thin Transaction State Machine

T0: Transaction Ready

- All:T0

  transaction request or response about connection
  send packet to the Connection register

  receive packet from the Connection register

  transaction indication or confirmation about connection

T1: SDU Send

- T0:T1
  transaction request of SDU Send
  send packet to the SDU register

  SDU transfer is completed
  send SDU Comp to the SDU management register

  transaction request of SDU Stop
  send SDU Stop to the SDU management register

  SDU transfer is not completed
  send packet to the SDU register

  Bus reset or unspecified error occurred
  T1:T2 transaction indication of error

  transaction response of SDU Send or SDU Stop
  send SDU Ready or SDU Reject to the SDU management register

  receive SDU Ready or SDU Reject from SDU management register

  transaction confirmation with SDU Ready or SDU Reject

  receive SDU Comp from SDU management register

  transaction indication of SDU Send with received data

  receive SDU Stop from SDU management register

  transaction indication of SDU Stop

T2: Error

- T0:T2
  Bus reset or unspecified error occurred
  T2:T2a transaction request or response about connection
  send packet to the Connection register

  receive packet from the Connection register

  transaction indication or confirmation about connection

- T1:T2
  internal event of reconnect complete

Figure 149 Thin Transaction state machine
Note that this state machine is intended to realize both inbound and outbound data transfers in one state machine. Therefore, the data receiving while data sending is not supported. To handle this, a node may have two state machines respectively for inbound and outbound data transfers, e.g.

**State T0: Transaction Ready.** This is the entry point of the Thin Transaction state machine. A data transfer process is initialized and ready for sending or receiving data.

**State T1: SDU Send.** A node is sending SDU. A node stays in this state until the SDU is sent completely.

**State T2: Error.** When bus reset or unspecified error is occurred, The Thin Transaction issues error indication.