Adobe Serial and Parallel Communications Protocols Specification

Adobe Developer Support

20 November 1992

Adobe Systems Incorporated

Adobe Developer Technologies
345 Park Avenue
San Jose, CA 95110
http://partners.adobe.com/

PN LPS5009

Contents

Adobe Serial and Parallel Communications Protocols Specification

1 Introduction

This document describes several protocols that can be used to communicate over a serial or parallel connection to a PostScript™ printing device, including *standard protocol*, *binary communications protocol* (BCP), and *tagged binary communications protocol* (TBCP).

The protocols described here are link-level protocols that are specific to serial and parallel communications channels. The protocols define special character sequences to indicate special control functions that are not logically part of the data stream.

Other communications channels provide such functions in entirely different ways, such as with special packet types in a local area network (LAN). When communicating via such channels or when saving a PostScript language program to a file, it is never appropriate to embed any of the control sequences described in this document. For an elaboration on this topic, see the section "Communication Channel Behavior" on pp. 74-75 of the *PostScript Language Reference Manual*, *Second Edition*.

2 Adobe Standard Protocol

The Adobe™ standard protocol is a simple communications protocol. Data is sent and received in ASCII. Several character codes are used directly by the communications driver for communications functions and are *not* passed through to the PostScript interpreter. These are listed in Table 1. Throughout this document, a control character is indicated by a ^ prefix.

Table 1 *Special characters in the standard protocol*

ASCII keyboard	*ASCII name*	*Value (hex)*	*Control function*
^A	SOH	0x01	Quote character for BCP & TBCP
^C	ETX	0x03	Generate an **interrupt** error
^D	EOT	0x04	End-of-file marker
^Q	DC1	0x11	XON in XON/XOFF flow control
^S	DC3	0x13	XOFF in XON/XOFF flow control
^T	DC4	0x14	Job status request
^[	ESC	0x1B	Start of end-protocol for TBCP

In Table 1, the ^A and ^[are only actually recognized by the standard communications protocol when the tagged binary communications protocol also exists, since they are used to begin and end the tagged binary communications protocol.

In the standard protocol, there is no way to ‘quote’ the reserved characters (in order to pass them through to the PostScript interpreter), nor is there any way to transmit characters in the ‘high ASCII’ range (128 to 255) if the high order bit is used for parity (odd, even, space or mark). However, this causes little difficulty in normal use since the standard PostScript language character set consists entirely of printable characters (plus the *space*, *tab*, and *newline* characters; see section 3.2.2, “ASCII Encoding” of the *PostScript Language Reference Manual, Second Edition* for more information). The language itself provides a means for encoding arbitrary characters in strings via the octal ‘\nnn’ escape sequence. True binary data, such as images and encrypted programs, must be transmitted in the hexadecimal (base 16) encoding or the ASCII base-85 encoding (*Level 2 only*) when using the standard protocol. Binary data may be transmitted on serial and parallel channels using one of the binary protocols described later in this document.

The standard protocol is supported on the RS232 and centronics channels of all devices.

2.1 Definition of Newline

The characters carriage-return (decimal ASCII 13) and line-feed (decimal ASCII 10) are also called *newline* characters. A carriage-return followed immediately by a line-feed are treated together as one newline by the PostScript interpreter when in the standard protocol mode. When a newline character is written to the standard output file, it is translated to the two-character sequence “carriage-return, line-feed”.

2.2 Enabling the Standard Protocol

Many printers have only the standard protocol for communicating via the serial and parallel channels. Therefore, it is not labelled the “standard” protocol in printer documentation, but merely discussed as “serial communications”. If it is the only protocol, then there is no need to “set” it. However, on printers that support other communications protocols, you may need to establish the standard protocol when you wish to use it.

Note *The proper way to enable the standard protocol differs between Level 1 and Level 2 printers. However, it is important to note that in both situations, the protocol change does not take effect until the end of the ‘setup’ job. Therefore, the protocol must be invoked as a separate job, rather than prepended to another PostScript language job. These methods are standard across all devices that support this protocol.*

On Level 1 printers, the communications protocol setting is associated with the current input/output mode. To return to the standard protocol from the binary communications protocol execute **setsoftwareiomode** in **statusdict** passing in the value of 0. For example,

```
%!PS-Adobe-3.0 ExitServer
%%Title: (Return to Standard Protocol - Level 1)
%%EndComments
%%BeginExitServer: 0
serverdict begin 0 exitserver
%%EndExitServer
statusdict begin
/setsoftwareiomode known {0 setsoftwareiomode}
end
%EOF
```

In Level 2, the standard protocol is set with the **setdevparams** operator setting the **Protocol** parameter to a value of **Normal**. Also, set the **Interpreter** parameter is set to the value **PostScript**. This will prevent configuration errors if the **Interpreter** parameter was previously set to something else. For example,

```
%!PS-Adobe-3.0
%%Title: (Set up Standard Protocol - Level 2)
%%EndComments
currentsysparams
/CurInputDevice 2 copy known {
   get                                % (%Device%)
   <</Protocol /Normal
   /Interpreter /PostScript>> setdevparams
}{
   pop pop
} ifelse
%EOF
```

3 Adobe Binary Communications Protocol

The Adobe binary communications protocol is an additional method of communicating between a PostScript printer and a host computer using an 8-bit wide serial or parallel channel. The protocol allows any of the 256 possible 8-bit values to be transmitted as data, but also allows certain characters to be used for specifying out-of-band control functions which may be handled synchronously or asynchronously by the communications driver. These control functions include flow control, status requests, aborting of jobs, and end-of-file markers.

The definition of the protocol is completely symmetric between the two communicating devices and does not assume that one system is a slave of the other. Binary means that all 256 byte values of 8-bit data may be sent and interpreted as data by either side. In addition, certain 8-bit values are reserved for out-of-band communications and control purposes. Although these last two sentences may seem contradictory, they express, in essence, the problem that the protocol solves—sending more than 256 distinct values encoded into a 256-valued alphabet.

Since any 8-bit value can be transmitted as data, this protocol can be used for sending PostScript language jobs that contain binary images or for sending to emulators data with any sequence of control characters. Since both PostScript language jobs and emulation jobs can be sent, this channel allows software switching between these modes without having to close and reopen the communications channel.

The binary communications protocol is supported on a few Level 1 printers and many Level 2 printers. To determine whether or not the protocol is available, applications should check the *PostScript Printer Description* (PPD) file. See the *PostScript Printer Description Files Specification* for more information.

3.1 Functional Description of the Protocol

The protocol is designed to be sent over channels that are logically 8 bits wide and normally support only a byte stream protocol rather than a packet protocol. The channel must be 8 bits wide because the fundamental unit that is being transmitted is a byte. The two most important examples of interest are asynchronous serial communications and 8-bit parallel communications. The binary communications protocol is appropriate only over channels that do not provide the same functions in other ways.

Although the definition of the protocol is completely symmetric, the particular functions of some of the out-of-band information are asymmetric in nature. The underlying assumption of the protocol is that “files” of data will be sent from one computer (referred to as the host) to the other (referred to as

the server) to be executed, each file as an individual “job”. The specific motivation for the protocol is files sent from a computer to a printer, where each file is considered a print job. One could envision other uses. For a more complete description of a job execution environment for interpreting files, see section 3.7.7, “Job Execution Environment” of the *PostScript Language Reference Manual, Second Edition*.

The out-of-band information (the non-data communications and control information) falls into two broad categories: those functions that are asynchronous with the data stream and those that are synchronous with the data stream. The protocol defines four out-of-band functions that are asynchronous and one that is synchronous.

The asychronous functions are:

- job status request. The server should respond immediately to job status request from the host by sending appropriate data back to the host. The syntax and semantics of the returned information are not specified by the protocol.

- job abort request. The server should respond immediately to job abort control from the host by terminating processing of the current job and flushing through the input stream until an end-of-file marker or end-protocol sequence is encountered. The server should proceed at that point with processing the end-of-file or end-protocol in the normal manner.

- XON flow control. The party receiving an XON may resume transmitting data that was blocked by a preceding XOFF. The XON and XOFF functions are present to support the well-established XON/XOFF flow control protocol used over asynchronous serial communications channels.

- XOFF flow control. The party receiving an XOFF should cease transmitting data as quickly as possible. It may still transmit asynchronous control functions, especially XOFF and XON. Flow control operates independently for each direction of data transmission.

The synchronous function is:

- end-of-file indication

As mentioned above, some of these functions are asymmetric in nature—it would not be expected, for example, that the server would send job abort request to the host.

3.2 Using the Binary Communications Protocol

The binary communications protocol is most useful for sending 8-bit data to printers over serial or parallel channels, without interrupting flow control functions. Data typically best suited for use with the binary communications protocol is binary image data, data output from a compression filter (*Level 2 Only*), or **show** strings that access characters encoded in the unprintable-ASCII range (below 32 decimal). An example is the IBM PC extended character set.

Using the binary communications protocol results in a significant performance advantage for users of serial and parallel ports. Using the standard protocol, the only way to send binary data over these channels is to use ASCII hexadecimal for image data (a 1:2 expansion of the data), the octal (‘\nnn’) notation for string data (a 1:4 expansion), or the ASCII base-85 encoding (4:5 expansion) (*Level 2 only*). The binary communications protocol offers a 1:1 transmission for most characters, with a 1:2 expansion for the few reserved characters.

Because the binary communications protocol is a device-dependent feature, it should not be used in Encapsulated PostScript (EPS) files or in print-to-disk files. In such cases, it is safer to avoid binary data or to use binary data with no escaping. Note that the binary communications protocol can be applied later to a job stream without needing any high-level knowledge about the job. Instead, it should be used only when the printer driver (or host application sending the job to the printer) can determine that it is connected directly to a printer. If your application does not have the ability to determine this, then the choice of using the binary communications protocol should be brought to the user-interface level.

3.3 Character Protocol

As mentioned in section 2, several character codes are used directly by the communications driver for communications functions and are not passed through to the PostScript interpreter.

Table 2 *Special characters in the binary communications protocol*

ASCII keyboard	*ASCII name*	*Value (hex)*	*Control function*
^A	SOH	0x01	Quote data character
^C	ETX	0x03	Generate an **interrupt** error
^D	EOT	0x04	End-of-file marker
^E	ENQ	0x05	(Reserved for future use)
^Q	DC1	0x11	XON in XON/XOFF flow control
^S	DC3	0x13	XOFF in XON/XOFF flow control
^T	DC4	0x14	Job status request
^\	FS	0x1C	(Reserved for future use)

To transmit these characters as data, they must be *quoted.* Quoting is done by replacing the character with a two-character sequence: the special character ^A (ASCII hex 0x01) followed by the character itself XORed with 0x40. For example, to send a byte with the value 0x14 (^T), the two-byte sequence 0x01 0x54 (^A T) is sent since ASCII T is the result of XORing ^T with 0x40. This method of quoting guarantees that whenever any of the special characters are received from the host computer, the control function is intended, regardless of whether the preceding character is a ^A.

The generation and processing of asynchronous control characters, therefore, can occur at a lower level than the generation and consumption of the data stream. In particular, on a host computer, the ^A quoting convention can be implemented by a user program while XON/XOFF processing is performed independently by the operating system.

All 8-bit values other than those of the special control characters listed in Table 2 are transmitted by simply sending the value.

After a ^A is received, the next character received that is not one of the special control characters must be the result of XORing one of the special characters with 40 hex. Receipt of any other character is considered an error in the input. Between the ^A and the XORed character, any number of the special characters can appear, except for special characters that are handled synchronously—^D and ^A. Receipt of ^D or ^A between a ^A and the XORed character is considered an error.

When one of the special characters arrives, unquoted, and it specifies no control function for the channel, the character is discarded. For example, if XON or XOFF is received and XON/XOFF flow control is not in use, XON or XOFF is discarded.

The characters ^E and ^\ currently specify no control functions. They are included among the characters that must be quoted in case new control functions are added in the future.

The following is a sample implementation for quoting characters in C.

```
#define UCHAR unsigned char

#define CTRLA           (UCHAR)0x01
#define CTRLC           (UCHAR)0x03
#define CTRLD           (UCHAR)0x04
#define CTRLE           (UCHAR)0x05
#define CTRLQ           (UCHAR)0x11
#define CTRLS           (UCHAR)0x13
#define CTRLT           (UCHAR)0x14
#define CTRLBKSL        (UCHAR)0x1C

/* QuoteByte quotes a character if necessary, using the Adobe Binary
 * Communications Protocol, and then writes the resulting bytes to
 * the printer by calling outbyte()
 */
void QuoteByte(c)
UCHAR c;
{
   switch (c)
   {
      case CTRLA:
      case CTRLC:
      case CTRLD:
      case CTRLE:
      case CTRLQ:
      case CTRLS:
      case CTRLT:
      case CTRLBKSL:
         outbyte(CTRLA);
         outbyte((UCHAR)(c ^ 0x40));
         break;
      default:
         outbyte(c);
   }
}
```

Table 2 shows the actions taken or the data put in the input buffer for byte sequences received from the channel by the communications driver.

Table 3 *Actions when bytes are received from hardware*

Read from hardware	*Data in input buffer*	*Type*	*Action*
^C	—	Asynchronous	Generate **interrupt**
^D	End-of-file mark	Synchronous	—
^E	—	—	—
^Q	—	Asynchronous	XON (flow control)
^S	—	Asynchronous	XOFF (flow control)
^T	—	Asynchronous	Printer status request
^\	—	—	—
^A A	^A	Synchronous	—
^A C	^C	Synchronous	—
^A D	^D	Synchronous	—
^A E	^E	Synchronous	—
^A Q	^Q	Synchronous	—
^A S	^S	Synchronous	—
^A T	^T	Synchronous	—
^A \	^\	Synchronous	—
^A <any other char>	Comm-Err mark	Synchronous	—
<any other char>	<char received>	Synchronous	—

3.4 Differences Between Standard and Binary Protocols

This section highlights some of the differences between the binary communications protocol and the standard protocol for serial and parallel channels.

- The binary protocol is entirely symmetrical. The control characters and the quoting conventions apply in both directions, although most of the control characters have no defined meaning in the printer-to-host direction.

- In contrast to the standard protocol, there is no mapping between end-of-line conventions in binary protocol. The end-of-line characters (CR, LF, or CR LF) sent by the host are received by the interpreter or emulator in the printer. The PostScript interpreter handles the different end-of-line con-

ventions in a uniform way, but a program that reads data from the channel directly (using **read** or **readstring**) receives the characters as sent by the host.

- Whatever output is generated by a PostScript language program (using **print** or **==**) is sent unchanged. Note that the standard PostScript language end-of-line (corresponding to the '\n' escape sequence in strings) which is normally carriage-return, line-feed, now simply becomes line-feed. This is especially noticeable in the **executive** mode of the interpreter.

3.5 Enabling Binary Communications Protocol

Before using the binary communications protocol, a driver must determine if doing so is appropriate—that is, communications are via a serial or parallel channel connected to a product that supports it. The binary communications protocol is never supported over communications channels that are inherently binary to begin with, such as Appletalk or SCSI. Additionally, the protocol is not available in all printers. To determine whether or not the protocol is available, applications should check the *PostScript Printer Description* (PPD) file. See the *PostScript Printer Description Files Specification* for more information.

Note *The proper way to enable the binary communications protocol differs between Level 1 and Level 2 printers. However, it is important to note that in both situations, the protocol change does not take effect until the end of the 'setup' job. Therefore, the protocol must be invoked as a separate job, rather than prepended to another PostScript language job. These methods are standard across all devices that support this protocol.*

On Level 1 printers, the binary communications protocol is associated with the current input/output mode. This is set using **setsoftwareiomode** in **statusdict**. For example,

```
%!PS-Adobe-3.0 ExitServer
%%Title: (Set up Binary Protocol - Level 1)
%%EndComments
%%BeginExitServer: 0
serverdict begin 0 exitserver
%%EndExitServer
statusdict begin
/setsoftwareiomode known {100 setsoftwareiomode}
end
%EOF
```

On Level 2 printers, the binary communications protocol is set by setting the **Protocol** device parameter to **Binary** with the **setdevparams** operator.

```
%!PS-Adobe-3.0
%%Title: (Set up Binary Protocol - Level 2)
%%EndComments
currentsysparams
/CurInputDevice 2 copy known {
   get            % (%Device%)
   <</Protocol /Binary>> setdevparams
}{
   pop pop
} ifelse
%EOF
```

This sets the device corresponding to the current communications channel.

Note *It is important to check the PPD file to confirm the existence of the binary communications protocol on a per-printer basis, because this is a device-dependent feature. More information on supporting device features can be found in Technical Note #5117, "Supporting Device Features."*

Also note that if the system parameter password (**SystemParamsPassword**) for the printer has been changed from the default, a **Password** entry will be required in the dictionary passed to **setdevparams**.

4 Adobe Tagged Binary Communications Protocol

This section describes the Adobe tagged binary communications protocol, a protocol for bidirectional binary communications between two computers.

The tagged binary communications protocol is very similar to the Adobe binary communications protocol though they are not compatible with each other. Note that neither protocol is an extension of the other. See section 3, “Adobe Binary Communications Protocol” for a description of the original binary protocol.

4.1 Functional Description of the Protocol

This section details the differences between the Adobe binary communications protocol and the Adobe tagged binary communications protocol.

Upon entering the tagged binary communications protocol, a sender should precede the data by a begin-protocol sequence. The characters that encode the begin-protocol sequence (see section 4.2, “Character Protocol”) are an illegal sequence in the Adobe binary communications protocol.

This choice was deliberate so that if one side is using the binary communications protocol and the other is using the tagged binary communications protocol, an error will be generated immediately.

The protocol may be thought of as a connection between the sender and the receiver. The sender begins the connection by sending the begin-protocol sequence. During the connection a sequence of files each separated from the next by an end-of-file marker is sent from the sender to the receiver. The sender terminates the session by sending an end-protocol sequence. Note that the last file in the connection should be followed immediately by an end-protocol sequence. If the last file is followed by an end-of-file marker which is then immediately followed by an end-protocol sequence an ‘empty’ file will be processed, which although in most environments should be benign, is not recommended.

A sequence of files sent to the server that does not start with a begin-protocol sequence (that is, it does not establish a connection) does not conform to the tagged binary communications protocol. The data may be encoded according to some other protocol, but the identity of that protocol is not part of the specification of the tagged binary communications protocol (see section 4.4, “Uses in a Language Switching Environment”).

The end-protocol sequence not only signifies the end of the file data but also indicates termination of the protocol (that is, it closes the connection between host and server) and that a change of interpretation context may occur. The protocol does not specify precisely the semantics of this last action. The

common intended use is to switch seamlessly from one job language to another. If further files are to be sent using the tagged binary communications protocol, a new connection must be established by preceding the files with a begin-protocol sequence. Typical usage of the tagged binary communications protocol on a printer which switches languages seamlessly is discussed in section 4.4 of this document.

4.2 Character Protocol

Table 4 lists the control characters that are treated as control functions rather than as data by the communications driver when they are received from the hardware:

Table 4 *Special characters in the tagged binary communications protocol*

ASCII keyboard	*ASCII name*	*Value (hex)*	*Control function*
^A	SOH	0x01	Quote data character
^C	ETX	0x03	Generate an **interrupt** error
^D	EOT	0x04	End-of-file marker
^E	ENQ	0x05	(Reserved for future use)
^Q	DC1	0x11	XON in XON/XOFF flow control
^S	DC3	0x13	XOFF in XON/XOFF flow control
^T	DC4	0x14	Job status request
^[	ESC	0x1B	Start of end-protocol sequence
^\	FS	0x1C	(Reserved for future use)

To transmit these characters as data, they must be quoted. Quoting is done by replacing the character with the two-character sequence ^A followed by the character itself XORed with 0x40. For example, to send a byte with the hex value 0x14, the two-byte sequence 0x01 0x54 is sent. This method of quoting guarantees that whenever any of the nine special characters are received from the hardware, the control function is intended, regardless of whether the preceding character is a ^A. The generation and processing of asynchronous control characters, therefore, may occur at a lower level than the generation and consumption of the data stream. In particular, on a host machine, the ^A quoting convention may be implemented by an application program or job spooler while XON/XOFF processing is performed independently by lower level communications code.

All 8-bit values other than those of the nine special characters are transmitted by simply sending the value.

After a ^A is received, the next character received that is not one of the asynchronous special characters must be the result of XORing 0x40 with one of the special characters or with ASCII character CR (0x0D). Receipt of any other character is considered an error in the input. The sequence ^A M (ASCII M is the result of XORing 0x40 with ASCII CR) indicates begin-protocol (see table 5). Between the ^A and the XORed character, any number of the asynchronous special characters may appear. Receipt of a synchronous special character between a ^A and the XORed character is considered an error.

The end-protocol sequence is the following 9 character ASCII sequence appearing within the quotation marks: “ESC%-12345X”. Note that ESC is the same as ^[. The receiver must parse for this sequence upon receipt of the initial ESC character. If the full sequence is not received all prior characters of the sequence received are passed through as data. If the full sequence is received it becomes an end-protocol sequence and is treated accordingly. Note that the end-protocol sequence may have asynchronous special characters interspersed, and this should not interfere with parsing the sequence.

Because of the requirement described above that the receiver must pass through as data all sequences that begin with ESC except for the end-protocol sequence, the sender may take one of two strategies in quoting the ESC character. It may quote all instances of ESC characters occurring as data. This is especially simple but may expand the data more than is desired. Instead it may choose to quote the ESC character only if it is followed by the remainder of the end-protocol sequence, “%-12345X”. An intermediate strategy might also be used. For example, the ESC character might be quoted only if it was followed by an ASCII %.

When one of the special characters arrives unquoted, and it specifies no control function for the channel, the character is simply discarded. For example, if XON or XOFF is received and XON/XOFF flow control is not in use, it is discarded. If a ^A M sequence (begin-protocol indication) is received after the initial one it will be discarded. The characters ^E and ^\ currently specify no control functions. They are included among the characters that must be quoted in case new control functions are added in the future.

Table 5 shows the actions taken or the data put in the input buffer for byte sequences received from the hardware. Note that ^[is the same as ESC.

Table 5 *Actions when bytes are received from hardware*

Read from hardware	*Data in input buffer*	*Type*	*Action*
^C	—	Asynchronous	Generate **interrupt**
^D	End-of-file mark	Synchronous	—
^E	—	—	—
^Q	—	Asynchronous	XON (flow control)
^S	—	Asynchronous	XOFF (flow control)
^T	—	Asynchronous	Printer status request
^\	—	—	—
^[%-12345X	TBCP end-protocol	—	—
^A A	^A	Synchronous	—
^A C	^C	Synchronous	—
^A D	^D	Synchronous	—
^A E	^E	Synchronous	—
^A M	TBCP begin-protocol	Synchronous	—
^A Q	^Q	Synchronous	—
^A S	^S	Synchronous	—
^A T	^T	Synchronous	—
^A \	^\	Synchronous	—
^A [	^[	Synchronous	—
^A <any other char>	Comm-Err mark	Synchronous	—
<any other char>	<char received>	Synchronous	—

4.3 Additional Points

- The tagged binary communications protocol is entirely symmetric. The control characters and the quoting conventions apply in both directions, although some of the control characters have no defined meaning in the server-to-host direction.

- The data that passes through the protocol is “raw”. The intent is that both sides can send pure binary data to each other. No data transformation (such as mapping line-feed to carriage-return, line-feed or vice versa) is carried out by the protocol. If these transformations are desired they must be done at a level above the protocol itself.

4.4 Uses in a Language Switching Environment

Although not part of the tagged binary communications protocol specification, certain printer software environments typify those which inspired the tagged binary communications protocol. The tagged binary communications protocol works well, and Adobe recommends its use, on printers that have a language independent software component and arbitrates which language will interpret an incoming job, such as Hewlett Packard's Printer Job Language (PJL). In such an environment, when the arbitrating code invokes the PostScript interpreter to handle a job using the tagged binary communications protocol, the job stream should have the form

```
^[ %-12345X  ^A M  <PS job encoded in TBCP>  ^[ %-12345X
```

In case the previous job left the software in an ill-defined state, the leading end-protocol sequence ensures that control returns to the arbitrating code, which is a well-defined state, before the next job begins. There are two possible cases of this:

1. The leading ^[%-12345X is the end-protocol sequence for a previous tagged binary communications protocol connection (not shown in the example), and is processed as described in the protocol specification.

2. The leading ^[%-12345X is not part of a tagged binary communications protocol connection but appears out of the blue. In this case, the protocol specification doesn't really have anything to say about what ^[%-12345X means. As a practical matter, the ^[%-12345X is received by the arbitrating code, which is responsible for handling it properly.

The ^A M causes the protocol to change from the default protocol (typically Adobe standard protocol) to the tagged binary communications protocol. Note this change occurs instantaneously without data loss or errors which could occur when a data stream encoded for one protocol arrives at a different protocol. When placed in the job stream, ^A M should immediately precede the first byte of a PostScript language job. The end-protocol sequence is overloaded in the sense that it causes the protocol to end, the job to end, and control to return to the arbitrating code, which may invoke a different language or return to the PostScript interpreter to interpret the next job.

As discussed in section 1, use of the tagged binary communications protocol (including the begin-protocol and end-protocol sequences) is appropriate only when using channels that do not provide the same functions in other ways. Control sequences such as ^A M and ^[%-12345X are not a part of the PostScript language and have no special meaning if encountered by the PostScript interpreter.

Appendix: Changes Since Earlier Versions Template

Changes since October 14, 1992

- In all appropriate tables, corrected the definition of ^C to be that it generates an **interrupt** error.

Changes since February 14, 1992

- In section 2.2, “Enabling the Standard Protocol,” added code to set the **setdevparams** parameter **Interpreter** to a value of **PostScript** in the Level 2 example.

- Fixed minor typographical errors.

Changes since August 8, 1991

- Converted Technical Note #5081, “Adobe Binary Communications Protocol” to this new specification, “Adobe Serial and Parallel Communications Protocols”.

- Added a section on the standard protocol.

- Added a section on the tagged binary communications protocol.

Changes since May 4, 1991 version

- Changed section “Enabling Binary Communications Protocol.” Applications should check a PPD file for existence of protocol, rather than how to enable it.

- Fixed minor typographical errors.

Changes since July 17, 1990 version

- Added section “Uses of this protocol.”

- Added example C implementation.
- Added section “Enabling Binary Communications Protocol.”

Index

S

T, U, V, W

X, Y, Z