The Display PostScript™ System

Adobe Systems Incorporated

Display PostScript NX Software Concepts and Facilities

Release 1.0

01 June 1993

Adobe Systems Incorporated

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

Display PostScript NX Software Concepts and Facilities

Any references to a “PostScript printer”, a “PostScript file,” or a “PostScript driver” refer to printers, files, and driver programs respectively that are written to support the PostScript language. The sentences in this book that use “PostScript language” as an adjective phrase are so constructed to reinforce that the name refers to the standard language definition as set forth by Adobe Systems Incorporated.

Contents

List of Tables

List of Figures

List of Examples

Display PostScript NX Software Concepts and Facilities

1 About This Manual

This manual tells application programmers how to bundle the Display PostScript NX software with an application program. The application can then display on an X Window System™ platform that doesn't contain the Display PostScript™ system extension to X. The sections of the manual are listed below.

- Section 2, "Overview of Display PostScript NX Software," introduces Display PostScript NX software and the components necessary to make it work.
- Section 3, "Configuring the System," describes possible configurations for running Display PostScript NX software on single or multiple hosts.
- Section 4, "Configuring an Agent," describes how to use special data files and command-line arguments to configure an agent.
- Section 5, "Starting an Agent," describes how to start an agent automatically or on the command line with *execnx*.
- Section 6, "Configuring the Client Library," describes how to use the *dpsNXargs.h* arguments and special environment variables to configure the Client Library.
- Section 7, "Client Library Concepts," describes special programming considerations that you must be aware of when incorporating Display PostScript NX software into your application.
- Section 8, "New Client Library Procedures," describes the Display PostScript NX procedures and header files that have been added to the Client Library.
- Section 9, "What Belongs In a Release," lists the Display PostScript NX components that you should include with your product release.

- Section 10, “Modifying Your Application’s Installation Utility,” gives advice on what to include in your installation utility to install your application with Display PostScript NX software.
- Appendix A, “Quick Start,” shows how to load and run Display PostScript NX software from a UNIX® shell.
- Appendix B, “Colormap Usage,” describes how to modify your *.Xdefaults* resource file to configure the use of the default X color map for Display PostScript applications.
- Appendix C, “How to Use the Pass-Through Information,” highlights the information that you should include in your end user documentation.

1.1 Related Software

The Display PostScript NX Software Developer’s Kit (SDK) lets a software developer who already has access to a Display PostScript development environment add Display PostScript NX capability to an application. If you do not have a Display PostScript development environment, you can obtain one by purchasing a Display PostScript SDK from Adobe.

1.2 Related Reading

An understanding of the standard Display PostScript extension to X is recommended prior to reading this manual. Refer to the following manuals in *Programming the Display PostScript System with X* (Addison-Wesley, 1993) for information.

- *Client Library Reference Manual*
- *Client Library Supplement for X*
- *Display PostScript Toolkit for X*
- *pswrap Reference Manual*

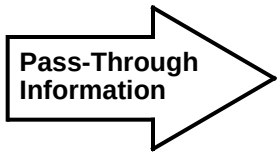
1.3 Typographical Conventions

The following typographical conventions are used in this manual:

Item	*Example of Typographical Style*
file or executable	*dpsclient.h*
variable, typedef	*ctxt* , *DPSContextRec*
code example	`typedef struct {`
C procedure	**DPSSetContext**
PostScript operator	**rectfill**
new term	“A *wrapped procedure* (wrap for short) consists of ...”

1.4 Other Notation

You should include some of the information in this manual in documentation for users of your application. The particular sections that contain information that you should include are indicated by the following icon. See Appendix C for more information.



Some path names mentioned in this document assume the directory that the files reside in can be described by the meta-variable *RELEASE*, with the addition of your vendor-specific name (top-level directory of the release), followed by the platform. For example, if Adobe System Incorporated’s evaluation tape were installed on */user/adobe*, the path would be:

```
RELEASE = /user/adobe/AdobeEvaluation-2/sparc
```

1.5 Glossary

The following terminology is used throughout this manual:

agent — The Display PostScript NX program that contains the PostScript interpreter. The agent has characteristics of both an X server (for example, it can accept network connections, process protocol requests, and multiplex simultaneous access) and an X client (for example, it sends drawing requests to an X server through Xlib).

application, or client	A program run by the end user, such as a paint program or word processor. The application does not need to run on the same machine as either the server or agent.
advertisement	Information placed by an agent on the X display's root window so potential clients can find the agent.
host	A computer that executes the program whose output appears on your display. A host is typically connected to other machines by a high performance network, such as Ethernet™. A host may be the workstation on your desktop, a large mainframe elsewhere in the building, or a supercomputer miles away. Hosts are identified by a network address and a name.
server, or X server	The part of the X Window System that provides the basic windowing mechanism for applications.
synchronization	Coordination between the application, the PostScript interpreter, and the X server to guarantee that all three are in a consistent state.
serialization	Transforming unordered disparate X and Display PostScript requests into a single stream of sequential requests on a single connection.

2 Overview of Display PostScript NX Software

This section provides an overview of Display PostScript NX software, including:

- How Display PostScript NX software works.
- Comparison of how Display PostScript NX software and the Display PostScript extension handle information flow and synchronization.
- Display PostScript connection policy used to determine if an agent should be used, and if so, which agent to use.
- How a port is selected for an agent that is started automatically by the Client Library.
- Requirements for a lock-and-key mechanism.

2.1 How Display PostScript NX Software Works

Display PostScript NX software is a host-based program that contains a PostScript interpreter. Instead of being integrated with the X Window System as in the Display PostScript extension to X, the interpreter communicates with the X server in the same way an application does—by means of X protocol. Because it does not depend on the presence of the Display PostScript extension, Display PostScript NX software lets your Display PostScript application display on any X server.

Design Features

The design features of Display PostScript NX software include the following.

- *Transparent support of the Display PostScript Extension to X.* Your application can use a single universal application programmer interface (API) that works, without additional programming, with systems that have either the Display PostScript extension or Display PostScript NX software.
- *Flexibility in configuration and deployment.* Display PostScript NX software lets end users configure their network or file system in different ways.
- *A variety of choices for starting the agent program.* Because an application developer or end user may want to configure and deploy Display PostScript NX software in different ways, multiple customizing choices are provided.

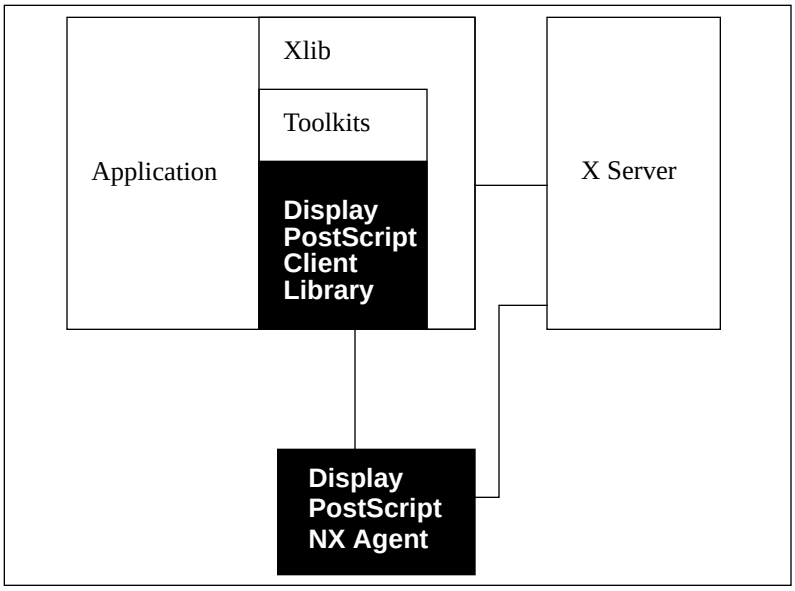
- *Protection of developer's investment.* Display PostScript NX software automatically implements a lock-and-key authorization system to guarantee that the agent purchased by an application developer serves only those applications that the developer specifies.

How the System Fits Together

The application sends PostScript language code to the agent, which executes the code and converts it into X drawing primitives. These drawing primitives are sent to the X server for rendering in the application window. The agent does not download large bitmaps; it draws using rectangles, tiles, stipples, colors, and fonts. The agent and application share the drawables and GCs specified at context creation or in the PostScript language program.

Because the application, agent, and X server can operate across a network, they may be distributed across different hosts or they may all run on the same host. Figure 1 shows how the system fits together. Each element is described below the figure.

Figure 1 *Functional elements of Display PostScript NX software*



- *Application.* The application must be linked with Xlib and with a version of the Display PostScript Client Library that is capable of using Display PostScript NX software.

- *Display PostScript Client Library.* The Display PostScript Client Library handles request management, status management, and event management for Display PostScript events. It also provides extensive support for synchronization and handling requests to an agent.

Note: *At this time, only the Client Library included in this release is capable of using Display PostScript NX software. In the future, other versions of the Display PostScript Client Library (such as those shipped by vendors of the Display PostScript extension) will support this software.*

- *Display PostScript NX Agent.* The Display PostScript NX agent is a separate process from the application. The agent contains the PostScript interpreter. The agent can be started by a Display PostScript application, another application, an end user, or a system administrator.

  The agent maintains a connection with the X server that is separate from the application's connection to the X server. To guarantee that X server resources are properly preserved, the agent makes a new connection to the X server for each application that connects to the agent.

- *X Server.* The X server provides a network-transparent window system. Applications communicate with the X server by using the X protocol. X servers are implemented on a variety of hardware platforms, including workstations, X terminals, and personal computers running X server software.

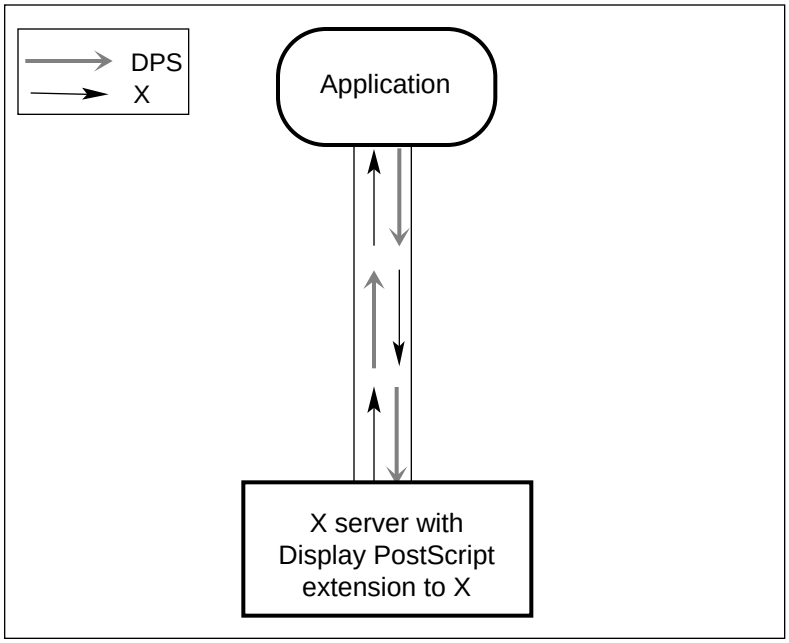
2.2 Information Flow and Synchronization

This section describes the way the Display PostScript extension to X and Display PostScript NX software handle information flow and synchronization. You must understand both in order to create a program that works with both.

Data Synchronization for the Display PostScript Extension to X

Figure 2 shows how information flows between an application and an X server with the Display PostScript extension. The dotted arrows in the figure indicate Display PostScript requests, replies, and events. The solid black arrows in the figure indicate X requests, replies, and events.

Figure 2 *Information flow in the Display PostScript extension*



X events and Display PostScript extension events are sent to the application on the same connection. Display PostScript requests and X requests are serialized and sent to the X server. If the request produces a reply, the X server sends the reply data back to the application.

Serialization imposes certain constraints in the way Display PostScript requests and X requests are processed. X requests are completed as soon as they are processed. In contrast, PostScript language code can take an unpredictable amount of time to complete. You must pay attention to these constraints so you know when to add code to synchronize requests. For example, if the PostScript interpreter renders a large scanned image into a pixmap, you must make sure that it is finished rendering before the X server copies the pixmap to a window. See Section 7.2 for more information on synchronization.

X and Display PostScript extension events are received by the application on the same connection. Output from a context, including text or data written to the context's output stream and return values from a wrap, is divided into separate chunks. Each chunk is small enough to fit in an X extension event. These events are sent back to the application, where they are assembled into a continuous output stream by the Client Library.

Your application uses X GCs and drawables with the Display PostScript system. The Client Library defines the GC values and drawable attributes that are used by the Display PostScript system. See *Client Library Supplement for X* for more information. Because the Client Library tracks changes to these X

resources for the Display PostScript extension, your application can use normal Xlib procedures, such as **XSetClipMask**, to make changes to GCs and drawables.

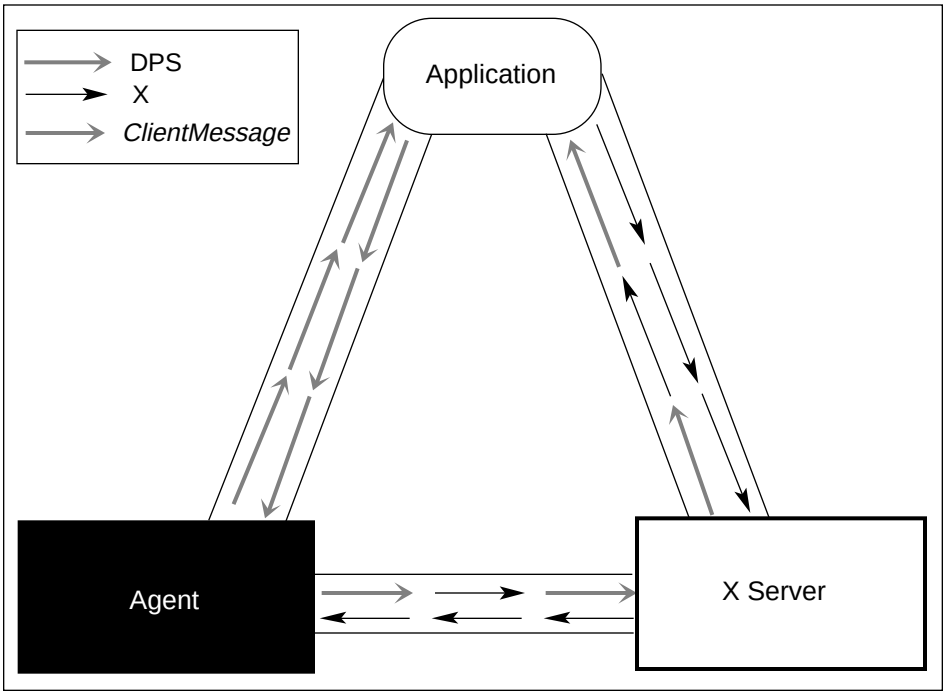
Data Synchronization for Display PostScript NX Software

While an application using the Display PostScript extension has only one network connection (X server), an application using Display PostScript NX software has two network connections (agent and X server) for each X server display session. The Display PostScript NX application sends X requests to the X server, and Display PostScript requests to the agent. To avoid the complexity of managing asynchronous events on two separate network connections, output from Display PostScript contexts is routed through the X server by means of *XSendEvent*. In this way, all asynchronous output, whether from X or from the agent, appears on the application's connection to the X server as X events.

Output from the Display PostScript context is divided into small chunks and sent as *ClientMessage* events. The agent sends these events to the X server, which in turn forwards them to the application through a private window known to both the Client Library and the agent. The Client Library assembles these events into a continuous output stream by overriding the low-level wire-to-event converter and intercepting the *ClientMessage* events before they are queued in the normal event queue.

Figure 3 shows how information flows between an application, the X server, and a Display PostScript NX agent. The dotted arrows in the figure indicate Display PostScript requests, replies, and events. The solid black arrows in the figure indicate X requests, replies, and events. The striped arrows in the figure indicate *ClientMessage* events.

Figure 3 *Information flow in Display PostScript NX software*



As in the Display PostScript extension, your application uses X GCs and drawables. For the most part, the Display PostScript Client Library automatically tracks changes to these resources. However, special consideration is necessary in cases where GC clips are created using **XClipRectangles** or **XSetRegion**. See section 7.1, “Handling Changes to the GC Clip,” for details.

Table 1 lists the Client Library and Xlib procedures you can use to handle information flow and synchronization.

Table 1 *Effects of synchronization and flushing*

Procedure	*Function*	*Display PostScript Extension Side Effects*	*Display PostScript NX Software Side Effects*
XFlush	Guarantees that pending X requests made by the application are sent to the X server. The application continues without waiting for confirmation.	Flushes Display PostScript requests.	None.
DPSFlushContext	Guarantees that Display PostScript requests made by the application are sent to the PostScript interpreter. The application continues without waiting for confirmation.	X requests are also flushed.	X requests are also flushed.

Table 1 *Effects of synchronization and flushing (Continued)*

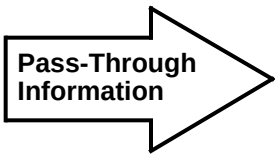
Procedure	*Function*	*Display PostScript Extension Side Effects*	*Display PostScript NX Software Side Effects*
XNextEvent, XPending, or any other Xlib calls that implicitly flush X requests.	Guarantees that pending X requests made by the application are sent to the X server. There are several Xlib and Intrinsics procedures for handling events which implicitly flush X requests before checking for events. Consult your Xlib and Intrinsics documentation to determine which procedures implicitly flush X requests.	Flushes Display PostScript requests.	None.
XSync, or X requests that return values.	Guarantees that pending X requests made by the application are sent to and processed by the X server. The application waits for confirmation before continuing.	Forces Display PostScript requests to be processed by the X server. Any PostScript language code contained in the requests, or already running in the interpreter, is not guaranteed to execute to completion.	None.
DPSWaitContext, or wraps that return values.	Guarantees that pending Display PostScript requests made by the application are sent to and processed by the PostScript interpreter. The application waits for the PostScript language code to be executed.	Guarantees that pending X requests made by the application are sent to and processed by the X server.	Guarantees that pending X requests made by the application are sent to and processed by the X server.

When you synchronize requests for the Display PostScript extension, you can use the Xlib procedures **XFlush** and **XSync** to flush and synchronize X requests. As you can see from Table 1, side effects cannot be relied upon, so you must use the procedures **DPSFlushContext** and **DPSWaitContext**. You must decide which procedure is appropriate for your application; some hints are given below.

Use **XFlush** or **XSync** (or requests that have replies) if you are trying to flush or synchronize X requests and you do not care about the status of Display PostScript requests. For example, you can use **XFlush** or **XSync** if:

- No Display PostScript requests are pending.
- Your application has just synchronized a Display PostScript request.
- Your application doesn’t care whether or not a Display PostScript request is synchronized because it plans to synchronize requests later.

Use **DPSFlushContext** or **DPSWaitContext** (or wraps that return values) if your program requires flushing or completion of Display PostScript requests. This includes calling **DPSFlushContext** before any Xlib or Intrinsics call that processes events and implicitly flushes pending requests. See section 7.3, “Handling Race Conditions Explicitly,” for more information.



2.3 Display PostScript Connection Policy

The Client Library follows a connection policy to determine whether an agent should be used, and if so, which agent to use. The default policy, described below, can be modified by the application developer or the end user.

1. Use the Display PostScript extension, if it is available.

   If an application tries to render on an X server that contains the Display PostScript extension, the extension will be used even if there is an agent available. You can force your application to connect to an agent regardless of the existence of the Display PostScript extension by setting the environment variable *DPSNXOVER* to the value “True”. This is not normally desirable except during debugging. For more information on *DPSNXOVER*, see section 6, “Configuring the Client Library.”

2. Connect to the agent specified by the application.

   An application may explicitly request a particular agent by using the **XDPSNXSetClientArg** procedure to set the argument *XDPSNX_AGENT* before creating a context. See section 8, “New Client Library Procedures,” for further details.

3. Connect to the agent that the end user specified through the *DPSNXHOST* environment variable.

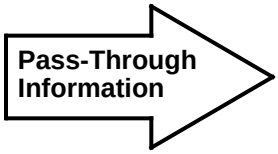
   An end user may specify an agent by setting the environment variable *DPSNXHOST*. If the end user specifies an agent and the Client Library is unable to establish a connection to it, context creation will fail. If *DPSNXHOST* is not set, step 4 is attempted. For more information on *DPSNXHOST*, see section 6, “Configuring the Client Library.”

4. Connect to an agent that is already servicing the X display.

   When an agent begins servicing an application on a particular display, it places an advertising property on the root window of that display so other applications can find it. The Client Library looks for agents that are advertising on the requested display. If the Client Library finds an available agent, it attempts to connect to it.

5. If automatic startup is specified, the Client Library starts an agent for the current display and connects to it.

   If the application has not set the **XDPSNXSetClientArg** argument *XDPSNX_AUTO_LAUNCH* to *True*, an agent will not be started and context creation will fail at this point. The Client Library attempts to start a new agent by searching the user's path for a *dpsnx.agent* file and invoking it. Applications may modify this default behavior by calling **XDPSNXSetClientArg** to specify a different agent name, a full path, and arguments to pass. See section 8, "New Client Library Procedures," for more details.



2.4 Selecting a Port

The following section describes how a port is selected for an agent that is started automatically by the Client Library. System administrators may need this information to change the default base port for TCP/IP transport. See section 5 for details on starting an agent.

The Client Library tries to start an agent on the port that was specified by the application when it called **XDPSNXSetClientArg**. If the application has not specified a port, the Client Library looks in the Internet services and aliases database for the entry

```
dpsnx  <port>/tcp
```

to determine whether there is a port assigned to it. Depending upon your network or system configuration, this database may be either the file */etc/services* or an NIS database. *port* is the IP port number assigned as Display PostScript NX software's base listening port. Its value is the base port in a range of ports that the Client Library and the agent expect to be available for their use. The specified network protocol must be "tcp". If the application has specified a port, but that port is already in use, the agent will not start. If there is no entry in the database, the Client Library tries to start an agent using a default port, which is hard-coded.

2.5 Lock-and-Key Authorization

Display PostScript NX software provides a lock-and-key authorization mechanism which guarantees that the agent you purchased serves only the applications you specify. Agents are locked and will accept connections only from applications that have an appropriate key. Your application doesn't have to do anything to take advantage of this feature.

For flexibility, keys are two-tiered. The primary tier is based on the application vendor's identification, usually the company name. This structure allows any or all applications developed by a particular vendor to use the

same agent. The secondary tier is a product number. This tier allows a vendor to organize products by agent, should the vendor choose to distribute agents with varying levels of functionality, or with varying configurations.

The default implementation uses the primary tier only. To enable the secondary tier, the application vendor must contact Adobe Systems Incorporated to obtain a reconfigured agent and Client Library.

The key is encoded in the Client Library. Any application linked with your copy of the Client Library will have your key and will work with your locked agent.

3 Configuring the System

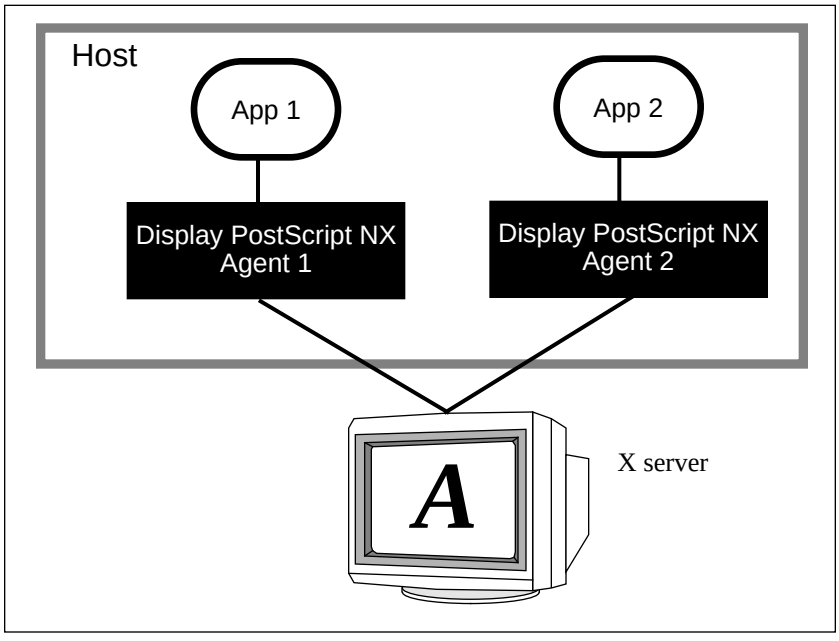
The following sections describe three possible configurations that you can choose from when deciding how to run the agent and application:

- One agent and one application run on the same host; the agent drives one display.
- Multiple applications and multiple agents run on one host; each agent drives one display.
- Multiple applications and one agent run on one host; the agent drives multiple displays.

3.1 One Agent per Application

Figure 4 shows the simplest configuration of an application, an agent and an X server. In this figure and those that follow, the direct connection between the application and the X server has been omitted for simplicity. See Figure 3 on page 10 for a complete diagram of the network relationship.

Figure 4 *One agent per application*

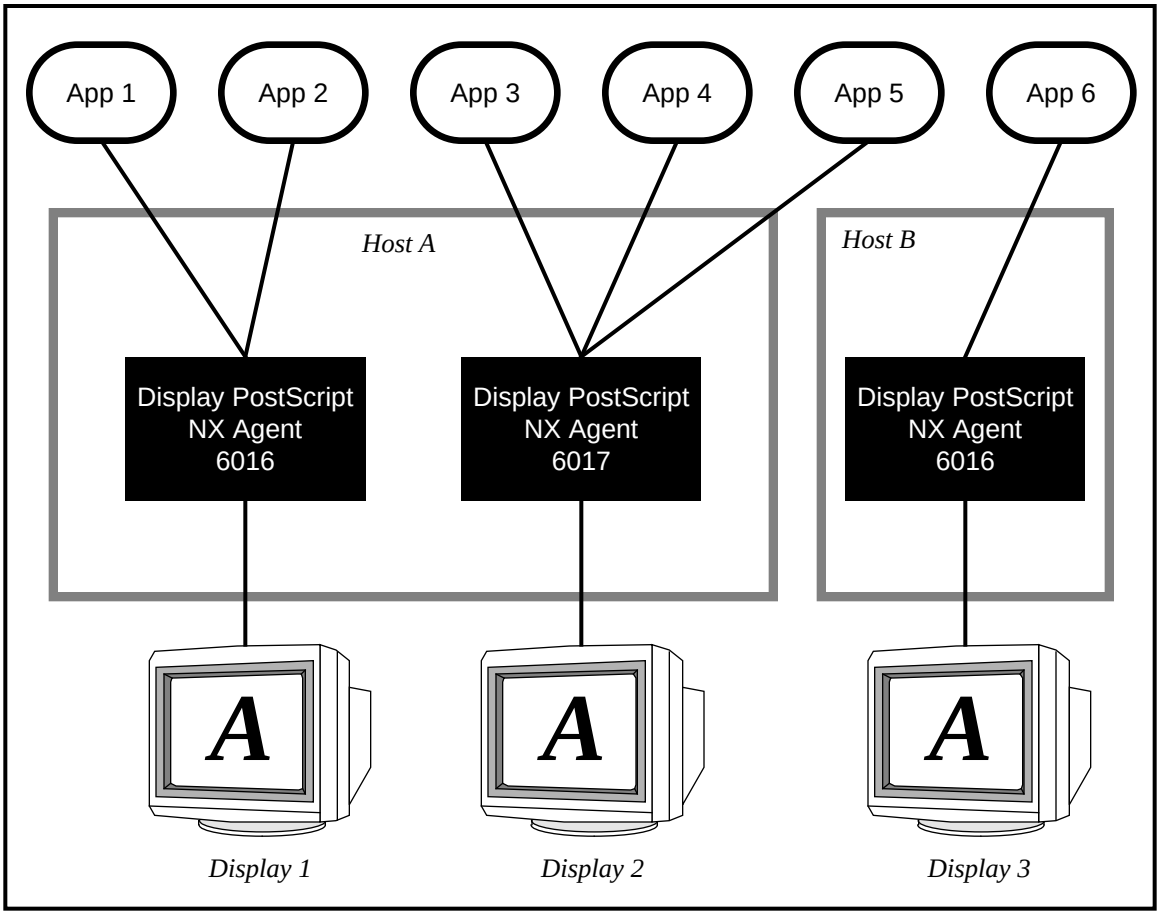


If Display PostScript NX software is set to automatically start an agent, the application and the agent will run on the same host. Otherwise, the application and the agent may run on different hosts. In either case, there is only one agent for each different application. For example, a word processor will use one agent and a spreadsheet will use a different agent. For multiple instances of the same application, there is only one agent. For example, two copies of the same word processor will use the same agent.

3.2 One Agent per Display

Figure 5 illustrates a more complex configuration; this configuration has the potential of multiple applications using multiple agents on a host.

Figure 5 *One agent per display*



Applications can run on the same host as agents, or on different hosts. If more than one agent runs on a host, each agent has its own port number. The number of ports that can be assigned on any given host is limited by the operating system on which you are running. The number of displays, agents, hosts, and applications shown in Figure 5 is only an example and is not meant to be restrictive. You may have many more of these elements.

One agent per display has several advantages.

- It makes the task of finding an agent easy. The application, which has already made a connection to the display that it wants to use, does nothing further. Instead, the Client Library looks at the properties on the root window of the display and finds a list of agents there. Only those agents currently servicing the display are on the list. In the configuration shown

in Figure 5, the list has exactly one agent. It doesn't matter which host the agent is on. The Client Library uses the advertising property list to make a connection to the right agent on the right host.

- This configuration closely models the Display PostScript extension. With the Display PostScript extension, there is only one PostScript interpreter per display. Likewise, in this configuration there is only one PostScript interpreter per display. This similarity has the best chance of yielding the same results between the Display PostScript extension and Display PostScript NX software.

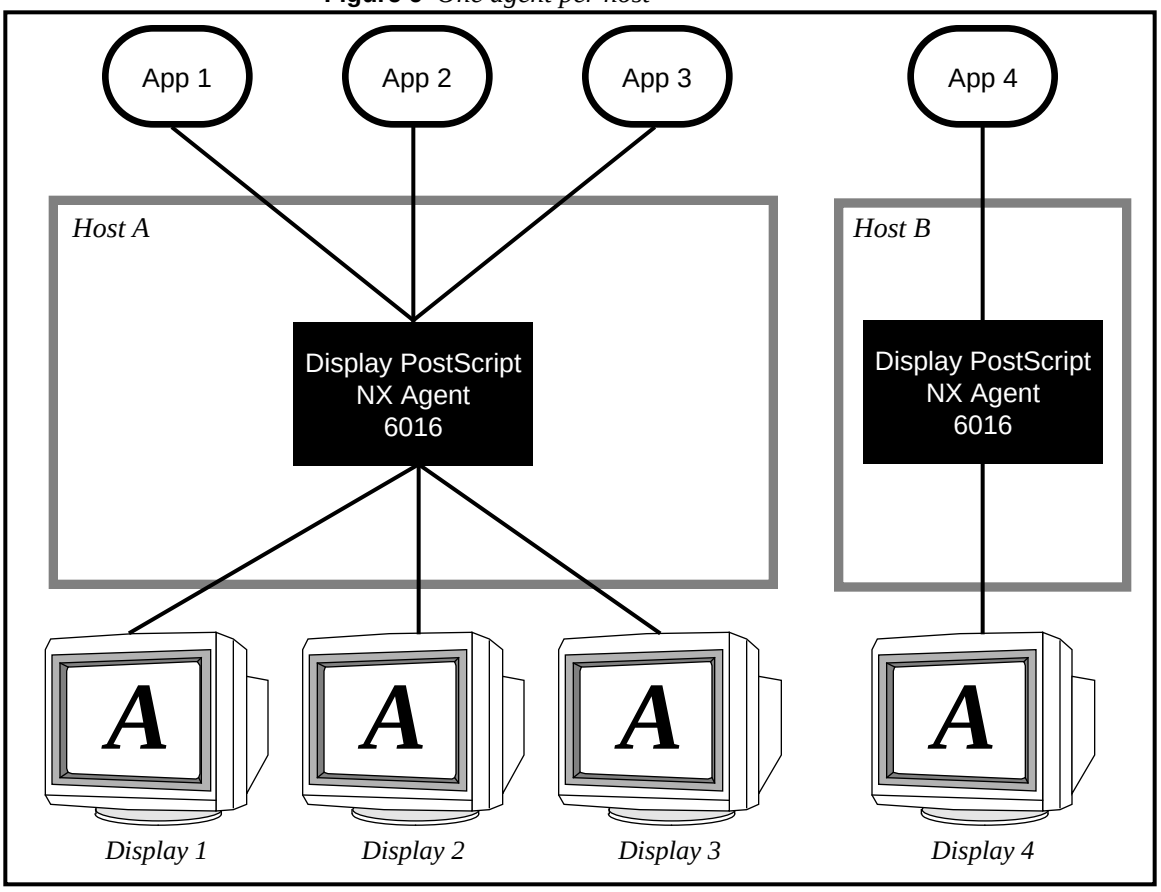
- This configuration is robust. If an agent crashes, only one end user is affected—the end user sitting in front of the display that the agent is servicing. Again, this organization parallels the Display PostScript extension organization. If the Display PostScript extension crashes, only the end user at the workstation is affected.

The main disadvantage of this configuration is that it can consume more host resources than other configurations. While most systems share executable program images, the dynamic data, which for Display PostScript NX software can be relatively large (working set of 1 to 2 megabytes), is not shared. A host with adequate memory and virtual memory management resources (paging and swapping space) should be able to handle an arbitrary number of agents, but overall system performance may be impaired.

3.3 One Agent per Host

Figure 6 illustrates another possible configuration. Applications may run anywhere. The number of applications, displays, agents, and hosts is just an example and is not meant to be restrictive, except that there is only one agent on any given host. Each agent drives one or more displays.

Figure 6 *One agent per host*



The main advantage of this configuration is that it minimizes the use of host resources. While the agent process itself may have a larger working set than in other configurations, most systems can manage a single very large process better than several large processes.

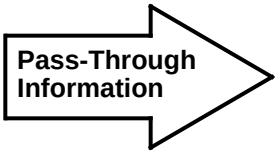
One agent per host has several disadvantages that correspond to the advantages of one agent per display. As long as all applications using the agent want to use the same display, the Client Library's use of advertising properties on the root window works. However, this default won't work for an application that wants to use a different display. The application must explicitly tell the Client Library to use the same agent for a different display. The application can override the default host or the end user can do this by setting the *DPSNXHOST* environment variable.

One agent per host cannot emulate the Display PostScript extension implementation as well as one agent per display. The main problem lies in the shared VM of the PostScript interpreter. With the Display PostScript extension, all applications displaying on a particular workstation can share information in shared VM but applications displaying on other workstations cannot. With one agent per host, applications showing on different displays may (but should not) share information in shared VM. Applications should avoid depending on this implementation inconsistency.

One agent per host is also not as robust as one agent per display. If the single agent driving twenty displays crashes (or the host it is running on crashes), all twenty end users are affected.

4 Configuring an Agent

You can configure an agent through data files, by starting an agent with command-line arguments, or by passing arguments in *dpsNXargs.h* to the Client Library procedure **XDPSNXSetAgentArg**. This section describes the data files and the command-line arguments. See section 8, “New Client Library Procedures,” for a description of the arguments that you can pass to **XDPSNXSetAgentArg.**



4.1 Data Files

Three types of data files are provided so that you or an end user can configure an agent: PostScript language resource files, the initial VM file, and font files. You should install these data files in your application’s installation directory and include the agent configuration in your application’s installation and configuration process. See section 10 for information on installation.

PostScript Language Resource Files

The agent uses its own resource database, known as the *PostScript language resources*, to locate files that describe and contain PostScript language objects. These files may be Adobe Font Metric files (AFM files), font outline files, PostScript language procedure sets, forms, patterns, encodings, or any named PostScript language object. See Appendix A in *Display PostScript Toolkit for X* for complete details about locating PostScript language resources.

Two resource files, *DPSNX.upr* and *DPSNXFonts.upr*, must be included with your application. *DPSNX.upr* can contain two resource classes: *DPSNXVM* and *DPSNXDebugLog.*

- *DPSNXVM* specifies the location of the initial VM used by the PostScript interpreter. *DPSNXVM* can be configured to have an absolute path or a relative path, as determined by the installation choices made by you or the end user. This resource is mandatory.

- *DPSNXDebugLog* provides resource definitions for the location of the debug log generated by an agent. The contents of the debug file depends on the debug option specified on the command line. If this class isn’t specified, the debugging log information will go to the standard error stream of the tty that the agent is started from. See Table 2 for a description of the command-line options. This resource is optional.

DPSNXFonts.upr specifies the location of the font resources that are installed with the agent. This file provides a central point from which to identify all the fonts available to the application and the agent.

Font Files

A number of fonts are included with Display PostScript NX software. Use the *DPSNXFonts.upr* file to define the location in which you choose to install these fonts. See section 10 for more information on installation.

Initial VM File

You must include the *dpsnx.vm* file with your application. This file contains the initial VM for the PostScript interpreter. It must not be changed by the end user and should have read-only attributes.

4.2 Command-Line Arguments

Table 2 lists the command-line arguments that you can use to configure the agent. Another program, such as *execnx*, usually starts the agent, so a user does not enter arguments on the command line. Instead, the application determines the agent configuration and constructs the command line programmatically. Your application may present a user interface that lets the end user specify command-line arguments for agent configuration. In the examples in Table 2, *execnx* is used to pass command-line arguments to the agent. Arguments specified after the – – indicator are passed to the agent through the automatic startup facilities in the Client Library (see section 5.2, “Starting an Agent with execnx”). Additional details about *–linger* and *–quantum* are discussed on the pages that follow the table.

Table 2 *Agent command-line arguments*

Command	*Description*
–debug n	Sets the debugging report level. All debugging information is sent to a log file, unless *–nolog* is specified. The *n* parameter, which should be in the range [0-9], specifies the amount of information to be reported. The higher the level, the more debugging information gets reported. Specifying 0 turns off reporting, except for warnings about exceptional conditions. Specifying a debug level greater than 6 generates large amounts of debugging data and is not recommended for normal use. Specifying a debugging level of 2 monitors basic connection and context creation activity. Level 0 is the default. For example: `execnx – – dpsnx.agent –debug 2`
–linger hh:mm	Forces the agent to keep listening for new connections. Normally, the agent exits when the last client disconnects. The *hh:mm* parameter indicates the amount of time the agent should wait for a new client after the last client disconnects. The *hh* parameter indicates the number of hours, the *mm* parameter indicates the number of minutes. The hours can be a single digit or omitted completely. The following example tells the agent to wait for 2 hours and 35 minutes. `execnx – – dpsnx.agent –linger 2:35`

Table 2 *Agent command-line arguments (Continued)*

Command	*Description*
–nolog	Sends debug log information to the standard error stream instead of the debug log file. For example: `execnx – – dpsnx.agent –debug 2 –nolog`
–psres dir	Specifies the default PostScript language resource directory for the agent. This directory is the value of "::" in the *PSRESOURCEPATH* environment variable, which can be specified by the user. See section 6 in this manual and Appendix A in *Display PostScript Toolkit for X* for more information. For example: `execnx – – dpsnx.agent –psres /usr/new/OurAppDir/DPS`
–quantum n	Sets the quantum, which is the number of operations a context is allowed to execute before it is forced to give up control. Higher numbers yield better context performance but poorer context scheduling; lower numbers have the opposite effect. For example: `execnx – – dpsnx.agent –quantum 300`
–stack n	Increases the amount of stack space allocated to each context by *n* bytes. If you are using a UNIX system with a modified kernel or your shell constrains the amount of stack space a process can have, you may need to add stack space to prevent the agent from crashing. For example: `execnx – – dpsnx.agent –stack 3000`
–sync	Turns on X synchronous mode in the agent for debugging. For example: `execnx – – dpsnx.agent –sync`

Controlling How Long an Agent Runs

Normally, an agent exits after the last application disconnects from it. If another application wants to use an agent shortly after the agent quits, it will have to start a new agent. You can make an agent stay active and wait for a new connection after the last application disconnects by using the command line argument *–linger* to specify how long an agent will wait for a connection. See Table 2 for details.

Context Scheduling

An application sends PostScript language code to a PostScript execution context to be executed. See *Client Library Supplement for X* for more information. Normally, an execution context runs until it needs more input, has output, or blocks for some reason. If none of these events occurred, a context could theoretically run forever and prevent other contexts from running. To prevent this from happening, each context is limited to a brief

execution period, or *timeslice*. The timeslice is controlled by a quantum number, which represents the maximum number of PostScript operators that a context can execute before giving up control.

The overall performance of an agent is sensitive to the quantum number. By default, the quantum number is set so that the performance is best when there is only one application using an agent. As the number of applications using an agent increases, the delay between one application's execution context being allowed to run and the next being allowed to run could become noticeable. For example, an application may run quickly for a short time and then freeze for a long time while other contexts are given a chance to run.

If several applications will routinely use an agent, consider experimenting with the quantum number until you get acceptable performance.

If you set the quantum number too low, the overhead of context switching is so great that performance suffers. If you set the quantum number too high, the agent's ability to respond to new connection requests and events is hampered.

5 Starting an Agent

An agent can be started automatically by an application or manually by using the program *execnx*. You can also start an agent by entering the executable name of the agent on the command line. However, this method is not recommended.

5.1 Starting an Agent Automatically

Your application can call the **XDPSNXSetClientArg** procedure to set the *XDPSNX_AUTO_LAUNCH* argument (defined in *dpsNXargs.h*). This argument configures the Client Library to start an agent automatically if an existing agent cannot be found. To take advantage of this feature, you must include the following code in your application:

```
XDPSNXSetClientArg(XDPSNX_AUTO_LAUNCH, True);
```

Note: You must call XDPSNXSetClientArg before you create the first Display PostScript context.

In addition to automatically starting an agent, your application may configure an agent by setting the command-line arguments listed in Table 2.

When *XDPSNX_AUTO_LAUNCH* is set to true, the Client Library is configured to automatically start an agent when needed. It will start an agent the first time the application tries to create a Display PostScript context.

Example 1 shows the code necessary to start an agent automatically. The agent's quantum is specified with command-line arguments. The transport and port are specified as well. Normally, the transport and port selection is managed by the Client Library. This example shows you how to add code to select the transport and port for exceptional cases.

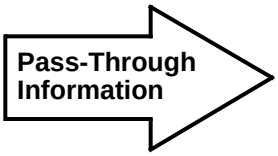
Example 1 *Configuring the agent with XDPSNXSetClientArg*

```
{
  char *quantum = “-quantum 300”
  char *agentArgs[2];
  DPSContext ctxt;

     agentArgs[0] = quantum;
     agentArgs[1] = NULL;
     XDPSNXSetClientArg(XDPSNX_LAUNCHED_AGENT_TRANS,
          XDPSNX_TRANS_TCP);
     XDPSNXSetClientArg(XDPSNX_LAUNCHED_AGENT_PORT, 3400);
     XDPSNXSetClientArg(XDPSNX_EXEC_ARGS, agentArgs);
     XDPSNXSetClientArg(XDPSNX_AUTO_LAUNCH, True);

ctxt = XDPSCreateSimpleContext(dpy, w, gc, 0, height,
       DPSDefaultTextBackstop, DPSDefaultErrorProc, NULL);
}
```

The name of the agent program is *dpsnx.agent*. When applications from multiple vendors are installed on the same system, this naming convention may cause a problem with lock-and-key authorization. For example, when your application tries to find *dpsnx.agent* on the search path, it may find another vendor’s version which will have the wrong lock for your application’s key. You can solve this problem by specifying an absolute path name in your code as follows.

```
XDPSNXSetClientArg(XDPSNX_EXEC_FILE,
      “/usr/new/OurAppDir/dpsnx.agent”);
```

You can also rename the agent to something other than *dpsnx.agent*; however, this approach is not recommended because some tools and applications provided by Adobe will not be able to find the agent.



5.2 Starting an Agent with execnx

execnx can be used to start a Display PostScript NX software agent from the command line. *execnx* lets you assign an agent to a specific display, ensures that the Display PostScript system is available before starting an application, and lets you configure an agent with any of its command-line arguments. To manually start an agent, invoke the *execnx* program from the command line. An example of the syntax is given below:

```
execnx[–display display_name:num[.screen]] [–new]
   [–port num] [–transport {tcp|unix|decnet}]
   [– –[agent_name] agent_options]
```

The command-line arguments for *execnx* are listed in Table 3.

Table 3 *execnx command-line arguments*

Command	*Description*
–display display_name:num[.screen]	Specifies the display for the agent to service. *display_name* refers to the name of the X server. *num* refers to the display number on that server. *.screen* refers to the screen number on that display. If *–display* is not specified, the *DISPLAY* environment variable is used.
–new	Overrides the default connection policy and starts a new agent. See section 2.3, “Display PostScript Connection Policy” for more information.
–port	Defines the network port to be used as the listening port for automatically started agents.
–transport	Sets the transport value to “tcp”, “unix”, or “decnet” for automatically started agents.
––[*agent_name*] *agent_options*	Specifies the file name of the agent executable and the command-line arguments to be passed through to the agent. *agent_name* is optional. See Table 2 for a complete list of the agent command-line arguments.

When *execnx* starts an agent, it also acts as the agent’s first client on the specified display. As a result, the agent places an advertising property on the given display and makes itself available to other applications.

By default, *execnx* will start an agent only for a display that is not already being serviced by either the Display PostScript extension or an existing agent. See “Starting Multiple Agents to Service a Single Display” on page 37 for an exception. If the X server for the targeted display contains the Display PostScript extension, *execnx* notifies the user that the Display PostScript extension is present. If the X server does not have the extension, but there is an agent servicing it, *execnx* will connect to the agent. If the display has an agent servicing it and the Display PostScript extension is present, *execnx* will not connect to the agent. To force *execnx* to start an agent, the environment variable *DPSNXOVER* must be set to “True”, and if an agent is present, the *–new* option must be used.

Starting an Agent to Service a Particular X Display

Use the following command syntax to start an agent for a particular X display.

```
execnx –display display_name:num[.screen]
```

execnx starts an agent to service the named display, assuming the host on which *execnx* is being executed has access rights to the named display. See the manual page for *xhost*(1) for more information. If the *DISPLAY* environment variable is already set to the display, you can omit the *–display* argument.

Starting Multiple Agents On a Single Host

The Display PostScript NX software requires a reserved set of TCP/IP ports so that multiple agents can run on a single host. (See section 2.4 for more information on selecting a TCP/IP port.) *execnx* automatically searches for an open port in this range so that it can assign a port to an agent.

Starting Multiple Agents to Service a Single Display

Multiple agents may service one display; it doesn't matter if the agents are all on the same host or on different hosts on the network. However, *execnx* will not start an agent to service the same display that another agent is already servicing, unless the *–new* argument is specified. Use the following command syntax to start an additional agent for a display.

```
execnx –display display_name:num[.screen] –new
```

This command line guarantees that *execnx* will start an agent for the named X display even if another agent is servicing it.

Starting a Specific Agent

By default, *execnx* attempts to start the first executable named *dpsnx.agent* that it finds on the user's search path. Multiple agents on the user's path that contain locks for different applications can lead to a problem because *execnx* will start the first agent that it finds, which may not be the agent for which your application has an authorization key. To specify an agent, you need to include the name of the agent on the command line. If you specify an agent name without a complete file path, *execnx* will search the current search path for that executable. The following command line guarantees that *execnx* will start the agent in *OurAppDir* instead of any other agent executables that might be on the user's search path.

```
execnx –– /usr/new/OurAppDir/dpsnx.agent
```

Starting an Agent on a Specific Port

execnx uses the command-line arguments *–port* and *–transport* to start an agent on a specific port. *execnx* will only use the port specified; it will not use the reserved set of TCP/IP ports described in section 4, "Configuring an Agent."

```
execnx –port 4761 –transport unix –new
```

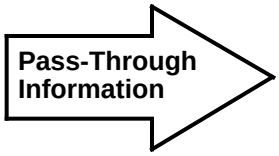
This command line will start a new agent that uses the UNIX domain port 4761. If the port is not available, the agent will fail to start.

Starting an Agent On a Different Host

execnx has no built-in functionality to remotely start agents. To start an agent on a specific host you must either remotely log on to that host and start the agent, or execute commands remotely on the host. For example, use the *rsh* command to run *execnx*.

6 Configuring the Client Library

You can configure the Client Library by setting special environment variables, or by using the procedure **XDPSNXSetClientArg** to set the arguments described by *dpsNXargs.h*. See section 8, “New Client Library Procedures,” for a complete discussion of the *dpsNXargs.h* arguments.



6.1 Environment Variables

This section describes the environment variables that you or an end user can set to override the default Client Library settings.

DPSNXHOST

Setting *DPSNXHOST* overrides the default location of the agent host and port that the Client Library will use to find an agent. If *hostname* is not set, the Client Library attempts to connect to the agent on the specified port. If *port* is zero, the Client Library attempts to connect to the agent on the default port. The syntax is:

```
<hostname><port separator><port>
```

where

hostname is either a name (for example, jumbo) or a number (for example, 131.30.0.212).

port separator should be “:” for TCP/IP and UNIX domain, or “::” for DECnet™.

port if *hostname* is a machine name, *port* specifies the IP port on which the agent is listening. If *hostname* is “unix” or is omitted, *port* specifies the suffix for the name of a UNIX domain socket.

DPSNXOVER

Use this environment variable if you want to force your application to use the Display PostScript NX software instead of the Display PostScript extension (for example, if you are debugging your application with Display PostScript NX software). This environment variable can be set to the string “True” or the string “False”; the capitalization does not matter. When *DPSNXOVER* is set to “True”, it instructs the Client Library to ignore the Display PostScript extension and use Display PostScript NX software instead.

PATH

This environment variable defines the current search path for executable programs. The Client Library expects to find the agent program in the current search path.

If you are distributing agent that contains a lock for your software, your application may have to redefine *PATH* so that it includes the location of the new agent first. However, a better way to do this is to get the exact location of the agent executable from your application's installation configuration data and use the **XDPSNXSetClientArg** procedure as described in section 5.1, "Starting an Agent Automatically," and section 10.4, "Using the Application Defaults File."

PSRESOURCEPATH

The environment variable *PSRESOURCEPATH* defines the list of directories that the agent and application will use to look for PostScript language resource files. *PSRESOURCEPATH* consists of a list of directories separated by colons. Two adjacent colons in the path represent the default directory. A typical example is:

```
::/usr/new/OurAppDir/DPS
```

The sample path above instructs applications to first look in the default place, then to search the directory */usr/new/OurAppDir/DPS*.

Note: Applications should not redefine or assume a value for PSRESOURCEPATH. This environment variable should be defined by the end user.

7 Client Library Concepts

The Client Library provided with Display PostScript NX software has the same API as the Client Library used with the Display PostScript extension to X. In most cases, to enable your Display PostScript application to take advantage of Display PostScript NX software, simply add code to configure the Client Library and the agent, as discussed in sections 4 through 6. In a small number of cases, additional programming may be required or desirable:.

- If your application changes the GC clip with **XClipRectangles** or **XSetRegion**, you must take special action to make your application work correctly with Display PostScript NX software. See section 7.1.
- If the automatic race-condition handling provided by the Client Library has a negative impact on the overall performance of your application, you may want to handle race conditions explicitly. See section 7.2 for an overview of race conditions and section 7.3 for advice on how to handle race conditions explicitly.
- If your application overrides the wire-to-event converter for *ClientMessage* events, you must take special action to make your override work correctly. See section 7.4.
- If the default font settings have a negative impact on the performance of your application, you may want to change the font settings. See section 7.5.

New procedures (**XDPSSyncGCClip**, **XDPSNXSetClientArg**, **XDPSNXSetAgentArg**, and **XDPSReconcileRequests**) have been added to the Client Library to handle these cases; they are documented in section 8.

7.1 Handling Changes to the GC Clip

The Client Library ensures that any changes to the GC values associated with a PostScript execution context are recognized by the context. However, GC clips used with Display PostScript NX software require additional attention. For more information on clipping and GC clipping in particular, see section 3.3 in *Client Library Supplement for X*.

The Client Library automatically handles GC clips by detecting changes in Xlib's GC cache. Unfortunately, GC clips created by using **XClipRectangles** or **XSetRegion** bypass the GC cache and go directly to the server. The Client Library has no way of automatically detecting these GC clips, so the application must take explicit action to keep the clip up to date.

Call **XDPSSyncGCClip** immediately after a GC clip is set with **XClipRectangles** or **XSetRegion**. This procedure ensures that any clip information that has changed is recognized by both the agent and X server.

You may also use this procedure after a clip mask is set. **XDPSSyncGCClip** flushes the specified GC and calls **XSync**. Call **XDPSSyncGCClip** no matter if you are using the Display PostScript extension or Display PostScript NX software.

7.2 Handling Race Conditions

In a *race condition*, the order in which two asynchronous requests are processed cannot be predicted; therefore the final outcome of the requests is unknown. Because Display PostScript NX software relies on separate network connections between the application, the agent, and the X server, race conditions can occur unless X requests and Display PostScript requests are synchronized. Two situations can cause race conditions to occur: Display PostScript requests followed by X requests and X requests followed by Display PostScript requests.

Display PostScript Requests Followed by X Requests

The first race condition happens when your application sends a Display PostScript request followed by an X request. This condition can impact applications running with either the Display PostScript extension or an agent. For example, a program issues a Display PostScript request to draw into a pixmap followed by an X request to copy the pixmap to a window on the display. Because the amount of time required to execute the PostScript language code is unknown, the X request may try to copy the image before the PostScript interpreter has finished rendering.

To ensure that the image is rendered correctly, the PostScript language code must be completed before the X request is processed. To accomplish this, use **DPSWaitContext** or other synchronization techniques. See section 4.8, “Synchronization,” in *Client Library Supplement for X* for more information.

X Requests Followed by Display PostScript Requests

The second race condition happens when your program sends an X request followed by a Display PostScript request. This condition impacts only applications running with Display PostScript NX software. For example, a program sends an X request to create a pixmap followed by a Display PostScript request to draw into the pixmap. Because the requests travel down different connections, the Display PostScript request may be processed first causing the agent to draw into a pixmap that doesn’t exist yet. Your application must ensure that the X request completes first.

The Client Library is configured to provide this synchronization. Each connection to the agent, which is identified by the X display handle for the X server, is set to the mode specified by the constant

XDPSNX_REQUEST_RECONCILE. This mode forces any X requests that preceded a Display PostScript request to complete before the Display PostScript request is processed.

By using *XDPSNX_REQUEST_RECONCILE* as the default mode, programs that were designed for the Display PostScript extension to X will work without change with Display PostScript NX software. However, because the Client Library does not know the nature of the X requests that preceded the Display PostScript request, or even whether there were any X requests, this method will perform less efficiently than an application that handles race conditions explicitly.

7.3 Handling Race Conditions Explicitly

The Client Library supports three request synchronization modes for use with Display PostScript NX software: *XDPSNX_REQUEST_RECONCILE*, *XDPSNX_REQUEST_XSYNC*, and *XDPSNX_REQUEST_BUFFER*. *XDPSNX_REQUEST_RECONCILE* and *XDPSNX_REQUEST_XSYNC* buffer requests and handle race conditions implicitly. *XDPSNX_REQUEST_BUFFER* only buffers requests, thus making it possible to handle race conditions explicitly. This section describes each mode, tells you why you might want to handle race conditions yourself, and gives code examples that show how to handle race conditions.

- *XDPSNX_REQUEST_RECONCILE* is the default mode and is adequate for most applications. This mode reconciles any outstanding X requests before a Display PostScript request is processed and does not cause the application to wait for confirmation. However, the PostScript execution context is paused while the X requests are reconciled, so PostScript code may take longer to execute.

- *XDPSNX_REQUEST_XSYNC* is provided for debugging or for situations when you do not want to pause the execution context in the same manner as *XDPSNX_REQUEST_RECONCILE*. This mode causes an **XSync** to be executed each time a Display PostScript request is made; **XSync** causes the application to wait until the X server confirms that all X requests have been processed. Forcing the application to wait may have a negative impact on performance.

- *XDPSNX_REQUEST_BUFFER* is provided for programmers who do not want to rely on the implicit handling of race conditions provided by the Client Library. For example, you would use this mode to improve the performance of the program. X and Display PostScript requests are buffered as they normally would be for the Display PostScript extension to X. Synchronization calls and X request reconciliation must be done explicitly at appropriate times, using the facilities discussed below.

If you decide to use *XDPSNX_REQUEST_BUFFER*, your application must handle all race conditions by doing explicit flushes and synchronization. Inadequate race condition handling may cause your application to hang, to render incompletely, or to lose information. You must ensure that X requests that are followed by Display PostScript requests, as well as Display PostScript requests that are followed by X requests, are handled properly. The following subsections discuss specific guidelines that need to be followed for synchronization; see “Examples of Implicit Flushing” on page 36 for guidelines on explicit flushing.

An Example of Handling Race Conditions Explicitly

To handle race conditions explicitly with *XDPSNX_REQUEST_BUFFER*, you must locate the places in your code where Display PostScript requests are made when the status of pending X requests is unknown. You must insert a call to **XDPSReconcileRequests** before the Display PostScript request. This task may not be as straightforward as it seems. For example, if you use a Display PostScript widget, you may not know when the Display PostScript requests are made. You must also handle all of the usual synchronization situations that you need to do for the other modes. For example, if you have X requests that depend upon the completion of PostScript code, you must insert a synchronization call, such as **DPSWaitContext**, before executing the X requests.

Example 2 shows an application before the proper synchronization code has been added. Example 3 shows the same code with the synchronization procedures **DPSWaitContext** and **XDPSReconcileRequests** added to handle race conditions. The additions are indicated with brackets. Example 3 resolves both types of race conditions.

Example 2 *Code without synchronization*

```
for (ln = big; ln > small; ln -= dsize)
  {
  dsize = (ln > 36.) ? ln * 0.1 : ((ln > 18.) ? 2. : 1);
  PSmoveto(8., .5*ln);
  PStransform(temp, 1.1*ln, &temp, &dy);
  ScrollDemoCanvas(-(int)dy);  /* Calls XCopyArea */
  PSselectfont(fname, ln);
  PSshow(text);
  XCopyArea(XtDisplay(demoCanvas), gb.canvas,
    XtWindow(demoCanvas), defaultGC, 0, gb.offset,
    width, height, 0, 0);
  }
ScrollDemoCanvas(50);  /* Calls XCopyArea */
XCopyArea(XtDisplay(demoCanvas), gb.canvas,
  XtWindow(demoCanvas), defaultGC, 0, gb.offset,
  width, height, 0, 0);
PSselectfont(fname, big);
```

Example 3 *Code with synchronization*

```
for (ln = big; ln > small; ln -= dsize)
  {
  dsize = (ln > 36.) ? ln * 0.1 : ((ln > 18.) ? 2. : 1);

  /*
  * You can't tell if X requests have been made before
  * this point, so call XDPSReconcileRequests to make
  * sure that all X requests have finished.
  */

  XDPSReconcileRequests(ctxt);
  PSmoveto(8., .5*ln);
  PStransform(temp, 1.1*ln, &temp, &dy);

  /*
  * Because PStransform returns at least one value,
  * Display PostScript requests are synchronized.
  */

  ScrollDemoCanvas(-(int)dy);  /* Calls XCopyArea */

  /*
  * Because ScrollDemoCanvas called XCopyArea, X requests
  * must be reconciled before doing Display
  * PostScript requests.
  */

  XDPSReconcileRequests(ctxt);
  PSselectfont(fname, ln);
  PSshow(text);

  /*
  * A Display PostScript request has been sent, and an
  * X request is about to be sent, so code needs to be
  * added to ensure that the Display PostScript request
  * completes first.
  */

  DPSWaitContext(ctxt);
  XCopyArea(XtDisplay(demoCanvas), gb.canvas,
    XtWindow(demoCanvas), defaultGC, 0, gb.offset,
    width, height, 0, 0);
  }
ScrollDemoCanvas(50);  /* Calls XCopyArea */
XCopyArea(XtDisplay(demoCanvas), gb.canvas,
  XtWindow(demoCanvas),defaultGC, 0, gb.offset,
  width, height, 0, 0);

/*
* An X request just occurred, so call
* XDPSReconcileRequest to make sure that the X
* request completes.
*/

XDPSReconcileRequests(ctxt);
PSselectfont(fname, big);
```

Limiting the Use of XDPSNX_REQUEST_BUFFER

Even by following the above techniques for synchronizing, it may not be possible, in all cases, to insert synchronization calls (for example, in complex callback procedure relationships when using Intrinsics-based toolkits). In these situations, try limiting the use of the *XDPSNX_REQUEST_BUFFER* mode to only those portions of your application where rendering performance is critical. You may do this by synchronizing all Display PostScript requests, and then changing the mode as shown in Example 4.

Example 4 *Limiting XDPSNX_REQUEST_BUFFER mode*

```
/* Upon entering a critical performance section */
DPSWaitContext(ctxt);
XDPSNXSetClientArg(XDPSNX_REQUEST_BUFFER, dpy);
....
/* Upon exiting the critical performance section */
DPSWaitContext(ctxt);
XDPSNXSetClientArg(XDPSNX_REQUEST_RECONCILE, dpy);
```

If you cannot get *XDPSNX_REQUEST_BUFFER* mode to work after trying these techniques, use the default *XDPSNX_REQUEST_RECONCILE* mode, as it always works.

Examples of Implicit Flushing

If you are using *XDPSNX_REQUEST_BUFFER* and you have already handled synchronization issues, you still need to handle flushing. You must flush Display PostScript requests whenever Xlib flushes X requests. Flushing requests is particularly important when using Intrinsics-based toolkits because flushing and event handling happen in non-obvious ways in these toolkits (see Table 1 for information on implicit flushing). To handle flushing, you must insert **DPSFlushContext** before the Xlib procedures that implicitly flush X requests. The following examples show the most common cases in which you will need to handle flushing.

Flushing Display PostScript requests before handling X events. In some cases, Xlib flushes X requests implicitly before handling events. Display PostScript requests should be flushed before calling an event procedure, as shown in Example 5.

Example 5 *Flushing Display PostScript requests before handling X events*

```
DPSFlushContext(ctxt);  /* flush before XPending call */
while (XPending(dpy)) {
   DPSFlushContext(ctxt);  /* flush before XNextEvent call */
   event = XNextEvent(dpy);
   ...
 }
```

Flushing Display PostScript requests when X requests return values. Display PostScript requests should be flushed before calling an Xlib procedure whose request returns a value, as shown in Example 6.

Example 6 *Flushing Display PostScript requests when X requests return values*

```
  DPSFlushContext(ctxt);
  name = XGetAtomName(dpy, atom);  /* flushes X requests */
```

Flushing Display PostScript requests before blocking. If your application will block for any reason, flush Display PostScript requests, as shown in Example 7.

Example 7 *Flushing Display PostScript requests for applications that block*

```
DPSFlushContext(ctxt);
c = getchar(); /*read character, may block until available*/
```

7.4 ClientMessage Wire-to-Event Override

You must be aware of the way the Client Library handles wire-to-event procedures if your application uses them. The Client Library overrides the default *ClientMessage* wire-to-event converter in order to invisibly intercept application messages that represent output from PostScript language contexts sent by the agent. It is therefore important that you daisy-chain (that is, call the old wire-to-event procedure if the new one is not interested in the event) any override that your application uses. If you don’t properly daisy-chain the override, information will be lost. A *ClientMessage* that is not recognized as a Display PostScript NX event is passed to the original wire-to-event procedure.

7.5 Special Considerations for Fonts

Because Adobe font characters are rasterized in the agent and must be sent as bitmaps to the server before they can be rendered, the overhead per character is greater than the same operation in the Display PostScript extension. You can improve the performance, with certain costs to accuracy, by using special font options that allow the use of X screen fonts. Most X servers are optimized to display these fonts quickly. Display PostScript NX software can use normal X11 protocol to open an X screen font and display characters from it.

If you decide to increase the speed of display for small font sizes, you will notice that characters and words will have inaccurate spacing:

- Text may lose its alignment if it is aligned on the right or justified.
- Adjacent characters may touch and occasionally overlap by a few pixels.
- Some characters intended to be adjacent may be spaced so far apart that they will appear to belong to two separate words.

You can control the trade-off between speed and accuracy by using one of the following methods.

- *Changing all fonts.* By default, the agent favors accuracy over speed. You may change this option, relative to the connection between the application and a given agent, by calling **XDPSNXSetAgentArg** to set *AGENT_ARG_SMALLFONTS* to *AGENT_SMALLFONTS_FAST*. The agent will try to use X screen fonts according to the settings in the font dictionary.
- *Changing on a font-by-font basis.* By default, fonts are configured to substitute a suitable screen font when there is a screen font available whose size matches the size requested. You may turn this option off by setting values in the font dictionary. See *PostScript Language Reference Manual, Second Edition* for further details. You must set *AGENT_ARG_SMALLFONTS* to *AGENT_SMALLFONTS_FAST* to use this method.

Note: The set of X screen fonts is relatively small but includes three typeface families that are commonly used: Courier, Times-Roman, and Helvetica. These fonts are available in sizes ranging from 8 to 24 points on most systems.

8 New Client Library Procedures

Display PostScript NX software extends the Client Library by adding a header file, *dpsNXargs.h* and four procedures in the existing header file, *dpsXclient.h. dpsXclient.h* now includes the following procedures:

- **XDPSSyncGCClip**—synchronizes the clip in the Display PostScript device with the clip in the X GC.
- **XDPSNXSetClientArg**—sets arguments that alter the behavior of the Client Library.
- **XDPSNXSetAgentArg**—sets arguments that alter the behavior of an agent.
- **XDPSReconcileRequests**—guarantees that X requests are processed by the X server before subsequent Display PostScript requests are processed by the agent.

Each procedure is described below.

Note: You do not need to test whether the Display PostScript NX software is running before calling the procedures. It is safe to use these procedures with the Display PostScript extension to X. For the extension, the procedures will either do nothing or do a comparable operation.

XDPSSyncGCClip XDPSSyncGCClip(Display *dpy, GC gc)

Synchronizes the clip in the Display PostScript device with the clip in the X GC. All applications should call **XDPSSyncGCClip** after calling either **XClipRectangles** or **XSetRegion** to guarantee that the application will work with Display PostScript NX software.

dpy The *Display* handle that specifies the X server for the display.

gc The X graphics context whose clip changed.

XDPSNXSetClientArg XDPSNXSetClientArg(int arg, void *value)

Modifies the Client Library defaults specific to Display PostScript NX software by changing arguments specified in *dpsNXargs.h*. All of the values that can be set by **XDPSNXSetClientArg** have defaults; see Table 4.

arg One of a defined set of constants documented in *dpsNXargs.h*. See Table 4.

value Defined as *void **. If the value will fit in the size of a *void **, pass the value itself. Otherwise pass the value as a pointer coerced to a *void **.

XDPSNXSetAgentArg Status XDPSNXSetAgentArg(Display *dpy, int arg, void* value)

Sets arguments that alter the behavior of an agent associated with the X server. It is assumed that *value* is of the correct type.

dpy The *Display* handle that specifies the X server that the application displays on. The *Display* handle is used to look up the connection to the agent. The argument that you set affects only this connection.

arg One of a defined set of constants documented in *dpsNXargs.h*. See Table 4.

value Defined as *void **. If the value will fit in the size of a *void **, pass the value itself. Otherwise pass the value as a pointer coerced to a *void **.

XDPSReconcileRequests void XDPSReconcileRequests(DPSContext ctxt)

Guarantees that any X requests sent prior to the call are processed by the X server before any Display PostScript requests made after the call are processed by the agent. This procedure returns immediately. It does not cause the application to wait.

ctxt The Display PostScript execution context that is to be used in a Display PostScript request that follows.

Note: If you use the mode XDPSNX_REQUEST_XSYNC or the default mode XDPSNX_REQUEST_RECONCILE, you do not need to call XDPSReconcileRequests.

Table 4 *dpsNXargs.h constants used with XDPSNXSetClientArg*

Name	*Type*	*Default Value*	*Description*
XDPSNX_AGENT	char *	*NULL*	This argument specifies an agent to connect to. Applications can offer end users the ability to set this value. The format of this value is: `[hostname]:[:]port` where: *hostname* is either a name (for example, jumbo) or a number (for example, 131.30.0.212). If *hostname* is a machine name, *port* specifies the IP port on which the agent is listening. If *hostname* is "unix" or is omitted, *port* specifies the suffix for the name of a UNIX domain socket. If *port* is zero, the default port is used. This argument overrides the environment variable *DPSNXHOST.*
XDPSNX_AUTO_LAUNCH	Bool	*False*	Set this argument to *True* to have the Client Library automatically launch an agent if one cannot be found. If the application sets *XDPSNX_EXEC_FILE*, *XDPSNX_EXEC_ARGS*, *XDPSNX_LAUNCHED_AGENT_TRANS*, and *XDPSNX_LAUNCHED_AGENT_PORT*, those values will be used to start an agent.
XDPSNX_EXEC_ARGS	char **	*NULL*	Lets applications specify command-line arguments to pass to an agent that is automatically launched. The last pointer in the array should be *NULL* to indicate the end of the arguments. *Note: Transport and port arguments should be set using XDPSNX_LAUNCHED_AGENT_TRANS and XDPSNX_LAUNCHED_AGENT_PORT instead of through XDPSNX_EXEC_ARGS, so that the Client Library will know how and where to connect to the launched agent.*

Table 4 *dpsNXargs.h constants used with XDPSNXSetClientArg (Continued)*

Name	*Type*	*Default Value*	*Description*
XDPSNX_EXEC_FILE	char *	*"dpsnx.agent"*	Lets applications use an agent that is installed in a specific directory. If you bundle your application with Display PostScript NX software using the lock-and-key licensing option, your installation procedure must install the matching agent executable file. If your application automatically launches an agent, use this argument to point the Client Library to the proper executable file for the agent.
XDPSNX_LAUNCHED_AGENT_PORT	integer	*system specific*	Lets applications explicitly set the port that a new agent will use as its listening port. If the specified port is not usable, the agent will fail to start. If the port is not set, the agent will attempt to use the default port.
XDPSNX_LAUNCHED_AGENT_TRANS	integer	*system specific*	Lets applications set the transport protocol that a new, automatically launched agent is to use. If it is not set, the Client Library will use the most efficient transport. The value should be selected from the transport value constants *XDPSNX_TRANS_UNIX*, *XDPSNX_TRANS_TCP*, or *XDPSNX_TRANS_DECNET*.
XDPSNX_REQUEST_BUFFER	Display *	*NULL*	Configures the Client Library so that when a Display PostScript request is made, it is buffered as it is for the extension. See section 7.3 for full details. This option gives the Client Library the best possible performance. Use this option only if you handle race conditions explicitly in your application code.
XDPSNX_REQUEST_RECONCILE	Display *	*NULL*	Configures the Client Library so that when a Display PostScript request is made, buffered X requests are processed by the X server before a Display PostScript request is processed by the agent. This is the Client Library default mode. See section 7.
XDPSNX_REQUEST_XSYNC	Display *	*NULL*	Configures the Client Library so that when a Display PostScript request is made, it calls **XSync** on the specified display before sending the Display PostScript requests. **XSync** guarantees that buffered X requests will be processed by the server before the Display PostScript request is sent to the agent. See section 7.

Table 5 *dpsNXargs.h constants used with XDPSNXSetAgentArg*

Name	*Type*	*Default Value*	*Description*
AGENT_ARG_SMALLFONTS	int	*AGENT_SMALLFONTS_ACCURATE*	Configures the agent so that fonts with small sizes (8-24 points) are shown with accurate spacing at the cost of slow performance by using *AGENT_SMALLFONTS_ACCURATE*, or as fast as possible with potentially inaccurate spacing by using *AGENT_SMALLFONTS_FAST*. Even if *AGENT_SMALLFONTS_FAST* is used, the agent may not be able to satisfy this request if suitable screen fonts cannot be found. The default is *AGENT_SMALLFONTS_ACCURATE*.
AGENT_ARG_PIXMEM	int	*AGENT_PIXMEM_LIMITED*	Informs the agent about the availability of pixmap storage on the X server. If there is unlimited memory (for example, the X server has virtual memory), the application can specify *AGENT_PIXMEM_UNLIMITED* to improve performance. If memory is limited, the application should specify *AGENT_PIXMEM_LIMITED* to minimize the agent's use of pixmaps. If the X server does not have limited memory, but it does not have limitless memory, the application can use *AGENT_PIXMEM_MODERATE*.

9 What Belongs In a Release

You must include certain components in your product release in order to distribute and use Display PostScript NX software correctly. You must also omit some components in Release 1.0 from your product release, because you may not be licensed to distribute these components.

9.1 Required Components

You must include the following components in your release.

Required Data Files

- *dpsnx.vm*
- Type 1 fonts and associated AFM files

  Courier

  Courier-Bold

  Courier-BoldOblique

  Courier-Oblique

  Helvetica

  Helvetica-Bold

  Helvetica-BoldOblique

  Helvetica-Oblique

  Symbol

  Times-Bold

  Times-BoldItalic

  Times-Italic

  Times-Roman

  Utopia-Regular

  Utopia-Italic

  Utopia-Bold

  Utopia-BoldItalic

Required PostScript Language Resource Files

- *DPSNX.upr*
- *DPSNXFonts.upr*

Required Pass-Through End User Documents

- man pages: *dpsnx.agent* (1), *execnx* (1), *listnx* (1)

Required Executables

- *dpsnx.agent*
- *listnx*
- *execnx*

9.2 Optional Example Programs

If your product is aimed at software developers, it may be useful to include the optional example programs provided in this release in either source or executable form with your product. These examples show code changes specific to the Display PostScript NX software.

9.3 What Does Not Belong In a Release

Adobe prohibits redistribution of the following files or directories that are inappropriate or that contain licensed or proprietary items.

Prohibited Documentation

Display PostScript NX Software Concepts and Facilities is targeted towards the application developer who has licensed Display PostScript NX from Adobe. The information included in this manual is inappropriate for the end user.

Prohibited Executables

- *dpsnx.debug*

Prohibited Development Files

- *RELEASE/include/*.h*
- *RELEASE/lib/*.a*
- *source.me*

10 Modifying Your Application's Installation Utility

This section gives advice on how to modify your application's installation utility. The installation utility that is used to install your application should also install the agent and the data files that it needs. The goal is to make installation of Display PostScript NX software as easy as possible for the end user.

Sections 10.1 through 10.4 discuss installation issues particular to platforms that do not have a previously installed Display PostScript development environment; section 10.5 discusses installation issues particular to platforms that have an installed Display PostScript environment.

10.1 An Example of an Installation Utility

See *RELEASE/build/motifdemos/dpsclock* for an example of how an application can use the methods discussed in this section. Examine the following two files in the directory:

- *install.dpsclock* is a C shell script that configures and installs the application and agent as described below.
- *dpsclock.c* demonstrates how an application reads application defaults and uses them to configure the Client Library and an agent.

10.2 Where to Install Files

In general, an installation utility needs to know where to place the application, *dpsnx.agent*, and PostScript resource files.

Where to Install the Application

In the methods described for installing Display PostScript NX software, the installation utility should know where the application is installed and whether the location is supplied by the end user or by default. The installation utility uses this information to install the agent.

Where to Install the Agent

It is up to you to make sure that the *dpsnx.agent* file is properly installed by your utility. You do not need to provide the end user a choice of where to locate the agent. Instead, your utility can put it in the same place on every system. However, to allow your end users greater control over the installation process, you may give them the option of specifying the desired directory. Your utility may suggest a default location. You can install the agent in the same directory as the application. Alternatively, Adobe recommends */usr/bin/X11* as the suggested default.

The Client Library needs to know the location of the *dpsnx.agent* file to automatically start an agent. The recommended way to supply the location is to pass its absolute path name as the *XDPSNX_EXEC_FILE* argument with **XDPSNXSetClientArg**. See section 10.4.

Where to Install PostScript Language Resource Files

The agent cannot run if it cannot locate the PostScript Language resource files. Your installation utility should obtain the resource file location during installation and configure the application so that it can pass this location to the agent at run time. The application should use the *–psres* argument to pass the absolute path name of the directory containing the resource file by calling **XDPSNXSetClientArg** to set the *XDPSNX_EXEC_ARGS* argument. See section 4 for more information on these resource files and command-line arguments.

10.3 How to Install Required Components

Your installation must install all required components, including *dpsnx.agent* and all required PostScript language resource files (*dpsnx.vm, DPSNX.upr DPSNXFonts.upr,* outline font files, and so on). See section 9 for a detailed list of the required components.

The PostScript language resource files are contained in a single directory in the Display PostScript NX release (*RELEASE/lib/DPS*). This entire directory should be copied during installation unless one or more of the following exceptions is true.

- The platform that you are installing on requires PostScript language resource files (particularly fonts) to be installed in predefined places.
- Your application is knowledgeable about PostScript language resources so that your installation utility can selectively install resources.
- You have developed an alternative strategy for integrating PostScript language resources from various sources.

10.4 Using the Application Defaults File

There are several ways to configure your application so that it can find the agent and the PostScript resource files. The easiest method is to add two general X resource names to the application defaults file: *dpsnxAgentExec* and *dpsnxPSResDir.* There is no support for accessing these resources in the Client Library, but the example *dpsclock.c* provides sample code. Adobe recommends that these resources be used and named as indicated below.

dpsnxAgentExec is a string that specifies the absolute path name of the agent. Your installation utility should insert this entry into the application defaults file during installation. For example, if the user specifies that the agent should be installed in */usr/bin/X11*, the installation utility should append this line to the application defaults file:

```
*dpsnxAgentExec: /usr/bin/X11/dpsnx.agent
```

dpsnxPSResDir is a string that defines the absolute path name to the default PostScript language resource directory. This value is passed to the agent with the *–psres* argument. For example, if the user specifies that the PostScript language resource directory should be installed in */usr/lib/X11*, the installation utility should append the following line to the application defaults file:

```
*dpsnxPSResDir: /usr/lib/X11/DPS
```

The PostScript language resource files are all located in *RELEASE/lib/DPS*. Your installation utility should copy the entire directory; the name of the directory should remain *DPS*. For example, if the PostScript language resource files are to be installed as specified by the *dpsnxPSResDir* string above, after installation is complete, the files would be found in

```
/usr/lib/X11/DPS
```

10.5 Integrating with an Existing Display PostScript Environment

The advice given so far applies to platforms that do not have a preinstalled installed Display PostScript environment. However, it is also possible to install the Display PostScript NX software on a platform that has a preinstalled Display PostScript environment. Two strategies can be used: ignore the existence of the preinstalled environment, or merge the resource files with the preinstalled files.

The first strategy, which is the easier, ignores the existence of the preinstalled environment and installs Display PostScript NX software independently. If you use this strategy, you must select installation directories that do not conflict with or overwrite the preinstalled system. For example, on DEC ULTRIX™, do not install the Display PostScript NX PostScript language resource files in */usr/lib/DPS*, where the DECwindows™ PostScript language resources are stored. To avoid this problem, design your installation so that all of your application files are located in one directory, for example, */usr/new/OurAppDir*. Install Display PostScript NX software in subdirectories of that directory, for example, */usr/new/OurAppDir/DPS*.

The second strategy involves merging PostScript language resource files from the Display PostScript NX release with the preinstalled files. If the preinstalled system uses the PostScript language resources mechanism, it is

possible to merge the font files in the Display PostScript NX release with those already on the system by making new entries in the *.upr* files. See Appendix A and Appendix B in *Display PostScript Toolkit for X* for further details.

If the previously installed system is not based on the PostScript language resources mechanism (for example, early versions of the Display PostScript system on workstations), use the first strategy and install Display PostScript NX software independently.

Appendix A Quick Start

The procedure in Table A.1 shows you how to load and quickly start Display PostScript NX software from this release, so you can see if everything works. You need a host to run Display PostScript NX software and an X server to display. After completing the procedure and trying out some of the programs, read the Release Notes to guide you through the rest of the release.

Follow the steps shown in Table A.1.

Table A.1 *Starting Display PostScript NX software*

Step	*Action*
1	Install the release tape by entering the *tar* command at the UNIX prompt. (Don't type the %, of course). For example: `% tar -xvf /dev/nrst8` Your device name may be different from *nrst8*. See the UNIX manual page *tar(1)* for more information on extracting files.
2	Create a new xterm by entering the following at the UNIX prompt in an available xterm: `% xterm –sb –rw &` This xterm will be used to run the Display PostScript NX agent.
3	Login to the host machine on the xterm: `% rlogin <machine_name>`
4	If you are not running the C shell, at the UNIX prompt, enter: `$ csh`
5	Set the *DISPLAY* environment variable to the display name: `setenv DISPLAY <hostname>:0` You may use *unix:0* or *localhost:0* or *:0*.
6	Change to the directory of the release (in this example, *AdobeEvaluation-2*) and then change to the platform directory (in this example, *sparc*): `% cd AdobeEvaluation-2/sparc`
7	Source the *source.me* file. `% source source.me`

Table A.1 *Starting Display PostScript NX software (Continued)*

Step	*Action*

8	Use the *execnx* program to start a Display PostScript NX software agent with the following arguments:
	
	`% execnx – – –nolog –debug 2`
	
	These command-line arguments select an optimized version of the agent (*dpsnx.agent*) with file logging turned off and sets the debugging level to 2. The following output is an example of what will be printed to the xterm:
	
	`%% DPS Client Library Warning:`
	`*** FORCING DPS NX ***`
	`%% DPS Client Library Warning:`
	`Auto-launching DPS NX agent.`
	`!0S: STARTING [Thu Apr  8 15:01:33 1993].`
	`!2S: Started on port 6016.`
	`!2H: New connection = 5 is TCP`
	`!2H: Completing connection handshake for client ci:1,cu:1 ...`
	`!2D: XOpenDisplay(xisbest:0.0) ...  done, fd:6.`
	`!2U: Put up advertisement on dpy "xisbest:0.0". id = 0x2c00001`
	`!2H: ... handshake completed.  REPLY SENT.`
	`!2P: RHCreateContext(ci:1,cu:1)`
	`DPS NX agent successfully started with pid: 18212.`
	`This program will block now.`
	`Put it in the background to maintain the auto-launched agent`

9	Repeat steps 2-7 to create a second xterm window. You will use this xterm to run the applications.
10	To exit Display PostScript NX software, quit all Display PostScript applications and interrupt or kill the *execnx* process.

If the output shown in step 8 does not appear, take the following actions in the order shown:

1. Review the steps to be sure you've followed them accurately.
2. Try doing the steps again from the beginning.
3. Be sure your host has "permission" to display on your X server. See the *xhost*(1) manual page for more information.

If you're still unable to start the Display PostScript NX software, describe the problem as fully as possible in an e-mail message to Display PostScript Support at Adobe Systems:

`dps-support@adobe.com`

Your message should include any error messages you received.

Appendix B Colormap Usage

This section provides information that is similar to the colormap usage information presented in *Client Library Supplement for X* except that this information is intended for end users instead of developers. An end user can use this information to customize the way the Client Library uses the colormap. Read section 3.2.1 in *Client Library Supplement for X* for considerations important to a developer.

This appendix describes how to modify your *.Xdefaults* resource file to configure the use of the default X colormap for Display PostScript applications.

The Client Library checks the *RGB_DEFAULT_MAP* and *RGB_GRAY_MAP* properties on the root window of the display. If these properties have already been defined, the Client Library uses these for its colormap allocations. If these properties have not been defined, the Client Library will define them, using defaults appropriate for the default visual type. You can customize these defaults by modifying the resources in your *.Xdefaults* resource file. Each resource entry should be of the form:

```
DPSColorCube.visualType.depth.color: size
```

where

visualType is one of GrayScale, PseudoColor, or DirectColor.

depth is 1, 2, 4, 8, 12, or 24 and should be the largest depth equal to or less than the default depth.

color is one of the strings “reds”, “greens”, “blues”, or “grays”.

size is the number of values to allocate of that color.

These resources are not used for the static visual types StaticGray, StaticColor, or TrueColor. Specifying 0 for reds directs the Client Library to use only a gray ramp. This specification is particularly useful for gray-scale systems that incorrectly have PseudoColor as the default visual. Giving the Display PostScript system more colormap entries improves the quality of its rendering, but leaves fewer entries available to other applications since the default colormap is shared.

For example, to configure a 5x5x4 color cube and a 17-element gray ramp for an 8-bit PseudoColor screen, specify these resources:

```
DPSColorCube.PseudoColor.8.reds: 5
DPSColorCube.PseudoColor.8.greens: 5
DPSColorCube.PseudoColor.8.blues: 4
DPSColorCube.PseudoColor.8.grays: 17
```

These resources use 117 colormap entries, 100 for the color cube and 17 for the gray ramp. For the best rendering results, specify an odd number for the gray ramp.

Resources that are not specified take these default values:

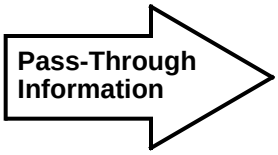
```
DPSColorCube.GrayScale.4.grays: 9
DPSColorCube.GrayScale.8.grays: 17

DPSColorCube.PseudoColor.4.reds: 2
DPSColorCube.PseudoColor.4.greens: 2
DPSColorCube.PseudoColor.4.blues: 2
DPSColorCube.PseudoColor.4.grays: 2
DPSColorCube.PseudoColor.8.reds: 4
DPSColorCube.PseudoColor.8.greens: 4
DPSColorCube.PseudoColor.8.blues: 4
DPSColorCube.PseudoColor.8.grays: 9
DPSColorCube.PseudoColor.12.reds: 6
DPSColorCube.PseudoColor.12.greens: 6
DPSColorCube.PseudoColor.12.blues: 5
DPSColorCube.PseudoColor.12.grays: 17

DPSColorCube.DirectColor.12.reds: 6
DPSColorCube.DirectColor.12.greens: 6
DPSColorCube.DirectColor.12.blues: 6
DPSColorCube.DirectColor.12.grays: 6
DPSColorCube.DirectColor.24.reds: 7
DPSColorCube.DirectColor.24.greens: 7
DPSColorCube.DirectColor.24.blues: 7
DPSColorCube.DirectColor.24.grays: 7
```

If none of the above defaults apply to the display, the Client Library uses no color cube and a 2-element gray ramp; that is, black and white.

Once the *RGB_DEFAULT_MAP* and *RGB_GRAY_MAP* properties have been defined by any application, all applications using that display will use the values of these properties, regardless of any customizations you may have made, until the X server is reset.

Appendix C How to Use the Pass-Through Information

Certain information presented in this manual should be included in your documentation for end users and in your documentation for system administrators. These concepts are marked throughout this manual with an arrow as “Pass-Through Information.”



You must decide which information is appropriate for your end users and incorporate it into your application’s documentation. The following sections highlight the information that is the most important. Adobe recommends that you at least include the information on starting an agent with *execnx* and colormap usage in your documentation.

C.1 Documentation for End Users

Display PostScript Connection Policy

If your application uses *XDPSNX_AGENT* and provides a user interface that lets your user select a host and port to connect to a specific agent, your documentation must describe the user interface.

If your application uses *XDPSNX_AUTO_LAUNCH* and an agent has been started, you should notify your end user that an agent has been started automatically. (The Client Library handles this by sending a warning message to the standard output stream by calling **DPSDefaultTextBackstop**). If *XDPSNX_AUTO_LAUNCH* is not specified and an existing extension or agent cannot be found, you should generate an error message that tells your user that the Display PostScript system is not available.

See “Display PostScript Connection Policy” in section 2.3.

Starting an Agent with execnx

Adobe recommends that you include information for end users on how to start *execnx* on the command line. Users may use *execnx* to do the following.

- Start an agent to service a particular X display.
- Override command-line arguments that are set by the application for automatically started agents.
- Start an agent if there are any lock-and-key conflicts.

See “Starting an Agent with execnx” in section 5.2.

Environment Variables

If you want end users to be able to set their environment variables to override the default settings in the application, you should include information that tells users how to set the variables.

If the user wants to use their own resource files in addition to the shared resources, you must include documentation that tells users how to modify the *PSRESOURCEPATH* environment variable.

Do not include the information on *PATH* if your application specifies an explicit path for the *dpsnx.agent* file.

See “Environment Variables” in section 6.1.

Colormap Usage

Adobe recommends that you include the information in this section in your end user documentation.

See “Colormap Usage” in appendix B.

C.2 Documentation for System Administrators

Selecting a Port

If a system administrator needs to change the default base port for the TCP/IP transport, you should provide information on how a port is selected.

See “Selecting a Port” in section 2.4.

Data Files

If a system administrator needs to change the location of the VM file, the debug log file, or add fonts to be shared by users of an application, you should provide information on how to change the PostScript language resource files.

See “Data Files” in section 4.1.

Starting an Agent With execnx

Adobe recommends that you tell system administrators how to start an agent on the command line with *execnx*. System administrators may use *execnx* to do the following.

- Specify where agents are run on a network.
- Override command-line arguments that are set by the application for automatically started agents.
- Start an agent on a specific port.
- Start multiple agents per display.
- Start multiple agents per host.

See “Starting an Agent With execnx” in section 5.2.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X