X PRINT SERVICE FUNCTIONAL SPECIFICATION

Revision 1.0

5/15/96

CDE/Motif PST

CDEnext

X PRINT SERVICE OVERVIEW

This Functional Specification describes, from the end user's and application developer's perspective, all the public components, APIs and GUIs that make up the X Print Service. The X Print Service enables X rendering to non-display devices such as printers and fax machines. For CDEnext, support for PCL and Postscript printers will be developed, and the architecture remains extensible to allow support for other non-display devices.

A companion book, The X Print Service Design Specification, serves as the blueprint document for the X Print Service, and addresses architecture, design opportunities and decisions, and other aspects of the X Print Service.

1.1 X Print Service Overview

The X Print Service from CDEnext is an architectural-level solution and sample implementation which allows X imaging to non-display devices such as printers. It is called the X *"Print"* Service because the primarily application of the technology will be towards printing, but the technology will in fact be applicable to a range of non-display devices. To date, print rendering technologies have evolved separately from display rendering technologies. The thrust of the X Print Service is to converge the evolution of these print and display technologies.

For example, today's X environment provides a number of APIs and technologies for rendering to a display, including:

- Xlib
- PEXlib
- X Imaging Extension
- OSF/Motif Toolkit
- Scalable Fonts

By retaining and supplementing these (and many more) standard APIs with one small print-specific API:

- libXp

The X Print Service will allow an existing X application to render against a printer in addition to traditional display devices (*note: the CDEnext sample implementation will not initially support all the X APIs*).

1.1.1 X Print Service Core Components

The X Print Service is made up of the following core components (proposed for standardization via the X Consortium):

- X Print Extension - a new X-Server Extension and corresponding X Print Extension Protocol.
- libXp - the X Print Extension Library which provides an API for applications to the X Print Extension Protocol.

CDEnext

- X Print "ddx" Drivers - ddx-level drivers for the X-Server that generate page description languages such as PCL and Postscript.
- Configuration Files and Defaults - configuration files that describe the capabilities of several printer models, and other X Print Server configuration files.

The X Print Service is enhanced by the addition of the following core components (proposed for standardization via CDEnext):

- libDtPrint - a library of print-specific GUIs tuned to several reference page-description-languages and printer models.
- dtpdm - also known as the Dt Print Dialog Manager, a daemon-like process that provides secondary printer-specific GUIs that handle specific printer and spooler setup tasks.

The following components are outside the scope of this project, and are not core component deliverables, but an attempt may be made to deliver them as "contrib" components:

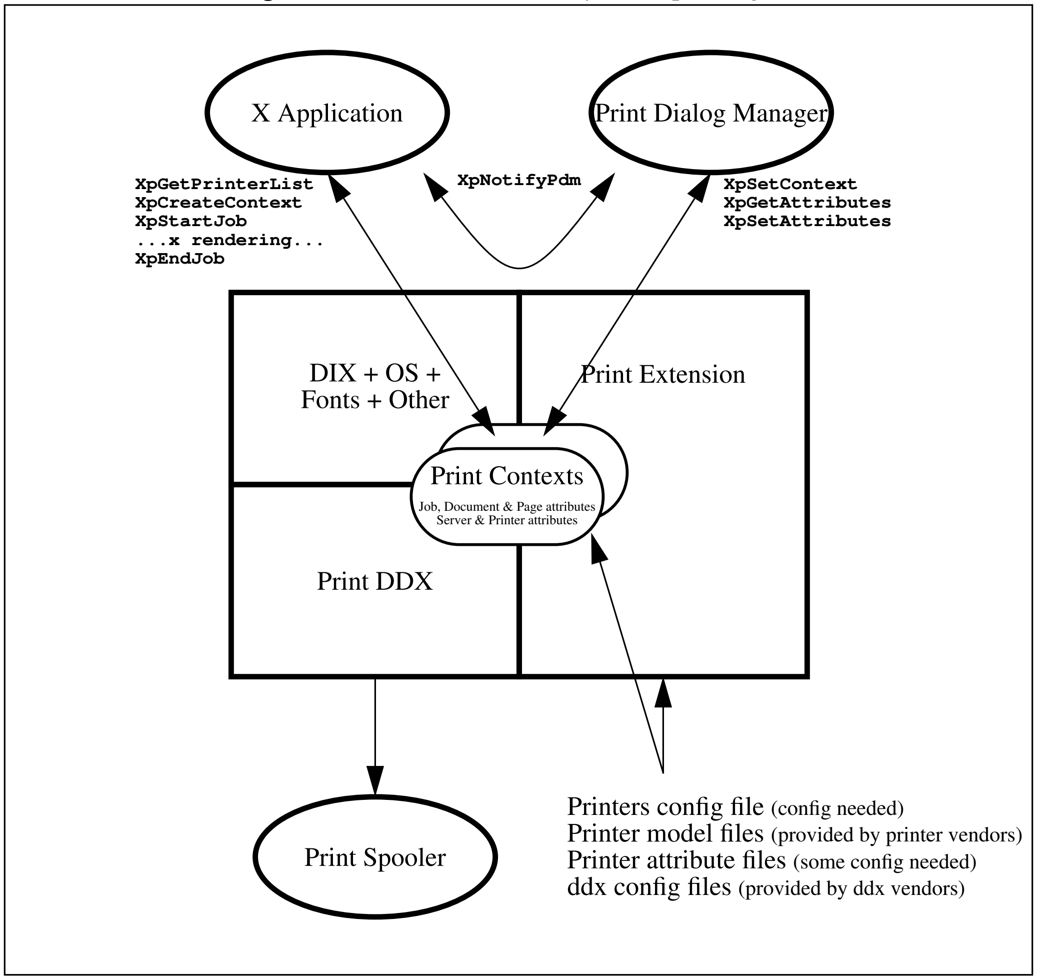
- A top-level application-embeddable dialog that handles the tasks of printer selection and generic printer setup (for example, number of copies), and offers hooks into the Print Dialog Manager.
- Installation and configuration scripts for specific vendor platforms.
- Auto-hosting Dt Print Dialog Manager protocol. Rather than depending on pre-configured security, an extension to the *PDM* Selection Protocol would allow on-the-fly "auto display-connection authorization" so that the *PDM* can display on the users *Video X-Server.*
- Some form of advanced color correction technology.

As the X Print Service was developed, keywords and concepts were borrowed from the abstract standard `ISO 10175`, and the subsetted standard and implementation represented by `POSIX 1387.4`, and the yet further subsetted implementation represented by `Palladium`. The X Print Service does not attempt to duplicate the functionality or APIs provided by any of these print subsystems, or by any other print subsystems such as `System V lp` or `BSD lp`. The X Print Service does attempt to "play with" these print subsystems however, in a least-common-denominator fashion, and is architecturally open enough to allow tighter binding to a specific print subsystem in the future.

1.1.2 X Print Service Key Concepts

The center of the X Print Service is the *X Print Server.* To an *X application*, it should look and behave like a regular *X-Server* with the following enhancements.

Figure 1-1. X Print Service Key Concepts Diagram



CDEnext

When the *X Print Server* starts, it may read a configuration file for instructions on which *print ddx drivers* to load, and which printer names to support. It may also read some "ddx dependent configuration files".

At this point, the *X Print Server* knows which printers to support, and has access to "printer model configuration files" that descibe the capabilities of the printer models. Parallel to the printer model configuration files are some "printer attribute configuration files" that can be modified if per-printer customization is desired.

When an *application* wishes to print, it can make a display connection to the *X Print Server* and ask to see the list of available printers via **`XpGetPrinterList`**. Once the *application* has selected a printer, the *application* can create and set a "Print Context" using **`XpCreateContext`** and **`XpSetContext`**. The "Print Context" is a critical concept, as it represents the embodiment of the printer selected - it is initialized by the *X Print Server* at **`XpCreateContext`** time, to contain the default capabilities of the printer, to contain the array of capabilities of the printer, to maintain the state of settings on the printer, to maintain the state of rendering against the printer, and to maintain the rendered output. The Print Context affects how the *ddx driver* generates its page description language (PDL), and how the PDL is submitted to a spooler. The Print Context may also affect fonts and other elements in the *dix layer* of the *X Print Server*. The most outwardly visible aspect of a Print Context are the "attribute pools" contained within, attributes that express and control: server, printer, job, document and page options. These attribute pools can be accessed and changed using **`XpGetAttributes`** and **`XpSetAttributes`**.

Because Print Contexts can be shared among processes, *applications* can enlist the help of a *secondary process* to manipulate print options in the Print Context rather than taking on the task directly. For CDEnext, the call **`XpGetPdmStartParams`** is being provided to enlist the help of the *Dt Print Dialog Manager*. By externalizing this task, new configuration dialogs and capabilities can be added without having to modify individual *applications*.

In most cases, the dialogs displayed by a *Print Dialog Manager* will be tuned to the capabilities of the corresponding *ddx driver*. It is possible to have multiple *Print Dialog Managers*, each one responsible for handling setup tasks for a different PDL.

Once the *application* has, with or without a *Print Dialog Manager's* help, set options within the Print Context, the *application* can make calls such as **`XpStartJob`** to delineate jobs, documents and pages within a sequence of normal X calls. Conceptually, a "job" is a collection of "documents", where each document is in turn a collection of "pages". When **`XpEndJob`** is called, the resuling PDL is sent to a print spooler, or can be retrieved by the *application*.

At this point, the end user should have on paper (or other non-display media) what was once only available on the display.

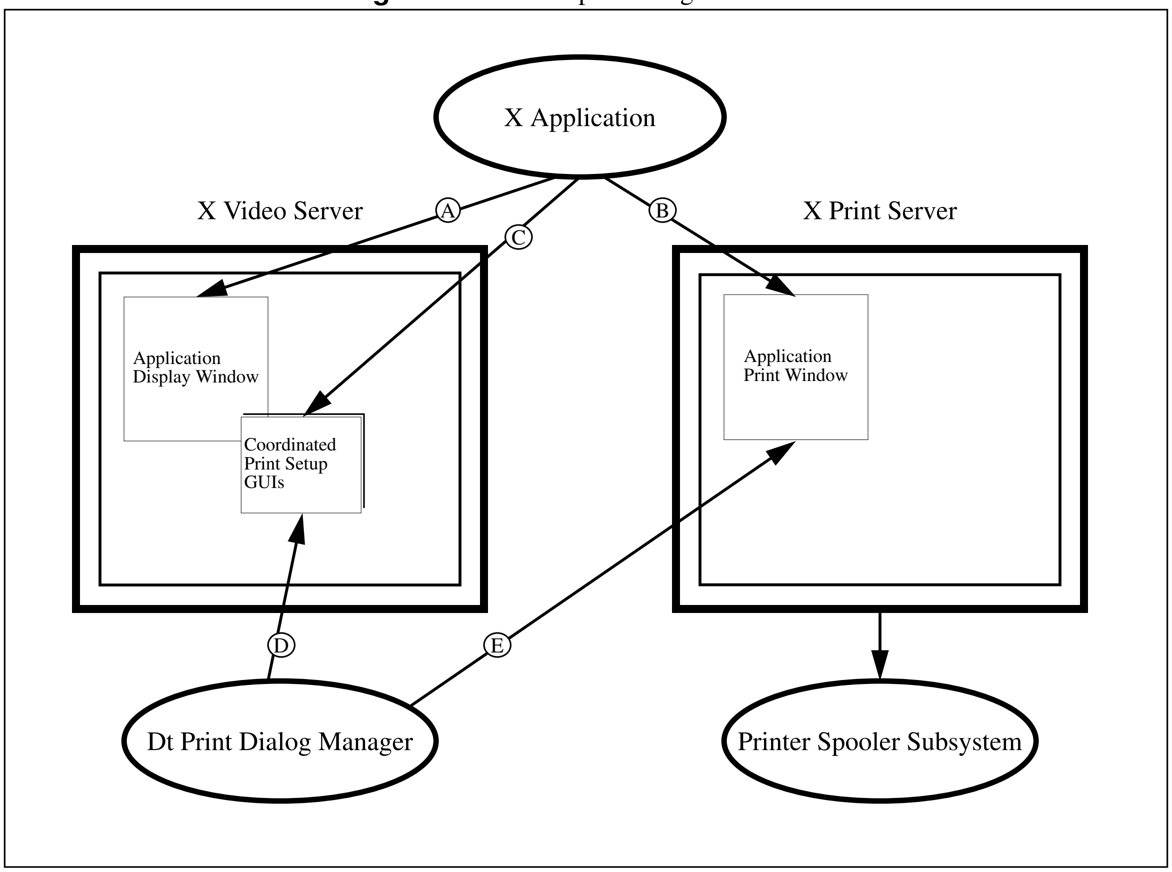
1.2 The Developer's/Integrator's View

The developer or integrator is the person who will modify an *X application* to use the X Print Service.

From the application's perspective, it can attach to one of two nearly identical X Servers (see figure points *A* and *B,* following diagram). The primary difference is that when connected to the *X Print Server*, additional calls can be made to delineate print "jobs", "documents" and "pages", and to create and modify a Print Context.

Conceptually, a "job" is a collection of "documents", where each document is in turn a collection of "pages". Depending on the print facilities underlying the *X Print Server* (for example, a print management system conforming to `POSIX 1387.4`), these delineations may be translated into tangible functionality.

Figure 1-1. Developer's/Integrator's View



CDEnext

A simple X application supplemented with some of the libXp routines might look like:

```
/*
 * Connect to the X Print Server
 */
pdpy = XOpenDisplay( printServerName );
/*
 * See if the printer "myLaser" is available
 */
plist = XpGetPrinterList( pdpy, "myLaser", &plistCnt );

/*
 * Initialize a print context representing "mylaser"
 */
pcontext = XpCreateContext( pdpy, plist[0].name );
XpFreePrinterList( plist );
/*
 * Possibly modify attributes in the print context
 */
attrPool = XpGetAttributes( pdpy, pcontext, poolType );
/* twiddle attributes */
XpSetAttributes( pdpy, pcontext, poolType, attrPool, XPAttrMerge );

/*
 * Set a print context, then start a print job against it
 */
XpSetContext( pdpy, pcontext );
XpStartJob( pdpy, XPSpool );
      /*
       * Generate the first page
       */
      pscreen = XpGetScreenOfContext( pdpy, pcontext );
      pwin = XCreateWindow( pdpy, pscreen, ..... );

      XpStartPage( pdpy, pwin, True );
      usual_rendering_stuff( pdpy, pscreen, pwin );
      XpEndPage( pdpy );
      /*
       * Generate more pages, and so on...
       */
      XpStartPage( pdpy, pwin, True );
      more_rendering_stuff( pdpy, pscreen, pwin );
      XpEndPage( pdpy );
/*
 * End the print job - the final results are sent by the
 * X Print Server to the spooler subsystem
 */
XpEndJob( pdpy );
XpDestroyContext( pdpy, pcontext );
```

CDEnext

1.2.1 Advanced Utilization of the X Print Service

To aid an application in printer selection and generic setup, and as a hook into the functionality provided by the *Dt Print Dialog Manager*, a primary or top-level X Print Service GUI (see figure point *C*) (*contrib component*) is provided in the library:

- libDtPrint

From the top-level GUI, a printer selection mechanism is provided that helps the user select from the available printers and their associated *X Print Servers* (e.g. *toms_laser* is available on *phub.hp.com:6*), though the *X Print Server* specification (e.g phub.hp.com:6) is by default hidden so that the more traditional printer-name based selection mechanism is presented. The top-level GUI also presents generic print setup options (e.g. *copies*) that can be modified once a printer is selected. Application specific options can also be embedded in this dialog, giving the appearance of a single unified print GUI.

For additional configuration, the *Dt Print Dialog Manager (PDM)* provides a second level of printer specific GUIs. An application *could* take on the task of printer configuration, or can easily offload the task to the *Dt PDM*. The executable is:

- dtpdm

The *dtpdm* is a standalone program that listens to and potentially serves applications attached to the *X Print Server* (see figure point *E*), and can be highly configured to the capabilities of each printer attached the *X Print Server.*

To give the appearance that the *Dt Print Dialog Manager* GUIs are also an integral part of the application, they are posted as transient windows (see figure point *D*) off the application. A typical service that the *Dt Print Dialog Manager* can provide is page size selection and resolution selection (e.g. select 8-1/2”x11” paper at 300dpi).

Using libXp API calls to the *X Print Server*, the application and *Dt Print Dialog Manager* can exchange configuration and option information. In some cases, the *Dt Print Dialog Manager* may work in close association with the *X Print Server DDX Driver* (e.g. X to PCL) to deliver higher level functionality and potentially tighter integration with the print spooler. For savy applications, it is possible to tie into the flow of configuration and option information, and make runtime changes.

1.3 The End User’s View

The end user is the person who will be using an X application that uses the X Print Service.

From the user’s perspective in a typical application, printing is done through a series of dialogs, the first one being initiated by selecting, for example, the pulldown menu <File><Print...>. The first dialog (contained in the application itself), will allow the user to select generic print options - such as the number of copies - as well as options specific to the application - such as page range. This initial print dialog will be comprised of two sections: generic print options, and application specific print options. The generic print options portion of the dialog will look and function identically across all applications. The application specific portion contains any options the application wishes to present in addition to the generic options.

1.4 The Printer Vendor's View

The printer vendor is the person or company that will wish to enhance the X Print Service so that it can support a new model of printer or page description language. Enhancements may be as simple as providing new printer model configuration files to providing a new *ddx driver* and corresponding *Print Dialog manager.*

The major points of enhancability within the X Print Service are:

- The *ddx driver* layer in the X *Print Server.* New ddx drivers can be added to support new page description languages, provide more capabilities, or provide tighter integration with a given printer model.
- The *Print Dialog Manager*, either as a new executable or enhancement of an existing Print Dialog Manager, to provide dialogs that expose highly printer specific options to the user and that communicates through the Print Context attributes with the *ddx driver.*
- The printer model files. These files describe the capabilities and defaults of printers based by model.

1.5 The System Administrator's View

The system administrator is the person who will configure and maintain the system processes and files associated with the X Print Service. The X Print Service has built-in fallback defaults for nearly everything, but can be highly configured to custom environments.

The X Print Service architecture has been designed so that support for specific page description languages and spooler subsystems is isolated to the *X Print Server's ddx layer* and a corresponding layer in the *Dt Print Dialog Manager.* With this architecture, support for new page description languages and spooler subsystems can be centrally added without having to reconfigure applications.

Support for specific types of printers and descriptions of the printer topology is also centralized in configuration files maintained by the *X Print Server.* Using libXp, the configuration information can be retrieved by applications and the *Dt Print Dialog Manager.*

The key areas of configuration and system administration are:

- X Print Service Startup - Deciding if a "per user session" or "global service" model of operation is desired. For the per user session model, the session manager (e.g. dtsession) could, if extended, start the *Print X Server* and *Dt Print Page Manager.* For the global session model, a *script* (or *program*) could be responsible for starting the *X Print Servers* and *Dt Print Dialog Managers*. And depending on the system administrator's security demands, the script may also distribute authorization information (e.g. MIT-MAGIC-COOKIE-1's) so that the servers can be accessed by applications (users).
- *X Print Server* Startup - Configuration files to control which printers are available.
- Attribute files - A collection of files that define the full range of capabilities of the printers accessed by the *X Print Servers* (e.g. 150, 300 and 600dpi supported), and default values (e.g. use 300dpi).
- Printer Model files - A collection of files typically supplied by a printer vendor to describe the capabilities of specific printer models (e.g. Laserjet 4si). These files will generally not require reconfiguration, but may be useful to reference when configuring files that describe the actual physical printers available (e.g. eliminate the selection of duplex printing because the printer's duplexer isn't working).

1.6 Functional Specification Chapters

The following documentation is provided in this functional spec:

- X Print Service Extension Library
- X Print Service Extension Events
- Application Print Dialogs
- Dt Print Dialog Manager
- X Print Service System Administration
- X Print Service Configuration Databases
- Fonts
- X Print Service Attributes
- X Print Server Driver Interface
- X Print Extension Protocol
- Glossary

1.7 Recommended End-User Documentation

From this Functional Specification, it is expected that something akin to the following End-User Documentation will be developed. The documentation will be in the form of a book, and in some cases, also available in true man(1) man pages.

- X Print Service - Programmer's Reference
    - X Print Service Overview
    - libXp Extension APIs + *man pages*
    - libXp Extension Events and Errors
    - libXp Extension Protocol
    - libXp Print Dialog Manager X-Selection Protocol
    - X Print Service Attributes + *possibly man pages*
    - libDtPrint APIs and GUIs + *man pages*

- X Print Service - System Administrator's Reference
    - X Print Service Overview
    - Configuring the X Print Server
    - Configuring the Print Dialog Manager
    - Configuring the X Print Service Attributes
    - Starting the X Print Service
    - Troubleshooting

- X Print Service - System User's Reference
    - X Print Service Overview
    - User Model
    - Desktop Integration
    - Configuring a User's Environment
    - Configuring the X Print Service Attributes
    - X Print Server + *man page*
    - Dt Print Dialog Manager + *man page*

- X Print Service - Technical Reference
    - CDEnext Raster-ddx capabilities/limitations in the X Print Server
    - CDEnext PCL-ddx capabilities/limitations in the X Print Server
    - CDEnext Postscript-ddx capabilities/limitations in the X Print Server
    - CDEnext Print Dialog Manager capabilites/limitations
    - Integrating New Print Drivers (ddx) into the Print X Server
    - Integrating New Print Dialogs into an Application and Print Dialog Manager
    - Understanding Fonts in the X Print Service
    - Understanding Supported Rendering Operations

X PRINT SERVICE EXTENSION LIBRARY

2.1 Overview

2.1.1 PURPOSE

These functions provide access to the X Print Protocol Extension to X. In addition, some convenience functions over the X Print Extension Protocol and Core X Protocols are provided that make it easier for an application programmer to use the X Print Service.

2.1.2 DESCRIPTION

The X Print Service Extension Library concentrates on print job, document and page management. The calls are:

- XpCreateContext
- XpSetContext
- XpGetContext
- XpDestroyContext
- XpGetScreenOfContext
- XpGetPageDimensions
- XpStartJob
- XpEndJob
- XpCancelJob
- XpStartDoc
- XpEndDoc
- XpCancelDoc
- XpPutDocumentData
- XpGetDocumentData
- XpStartPage
- XpEndPage
- XpCancelPage
- XpSelectInput
- XpInputSelected
- XpGetAttributes
- XpSetAttributes
- XpGetOneAttribute

- XpGetPrinterList
- XpFreePrinterList - convenience routine
- XpRehashPrinterList
- XpQueryVersion
- XpQueryExtension - convenience routine
- XpQueryScreens
- XpGetPdmStartParams - convenience routine
- XpGetAuthParams - convenience routine
- XpSendAuth - convenience routine
- XpSendOneTicket - convenience routine
- XpSetLocaleHinter - convenience routine
- XpGetLocaleHinter - convenience routine

2.1.3 DEPENDENCIES

The X Print Service is an extension to the Core X protocol, and cannot be used outside of the X environment.

2.1.4 ISSUES

2.2 XpCreateContext

2.2.1 Short Description

Creates and initializes a new print context.

2.2.2 Long Description

NAME

XpCreateContext - create and initialize a print context

SYNOPSIS

```
#include <X11/extensions/Print.h>

XPContext XpCreateContext(
   Display             *display,
   char                *printer_name)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
printer_name	The name of a printer on *display*. String encoded as `COMPOUND_TEXT`.

RETURN VALUE

`context_id` for the new print context.

DESCRIPTION

`XpCreateContext` creates a new print context that is initialized with the default printer attributes and other information available for *`printer_name`* on *`display`*. A print context maintains the printer name, print attributes, font capabilities, print (rendering) state and results, and is the object upon which the Xp calls act.

If the library fails to generate a new print context id, a value of `None` is returned, otherwise a print context id is always returned. If *`printer_name`* is invalid, a **`BadMatch`** will be generated later by the *X Print Server*.

A call to **`XpGetPrinterList`** will return a valid list of values for *`printer_name`*. All printer name values in the X Print Service are encoded as `COMPOUND_TEXT` (of which the ISO-8859-1 code-set is a proper subset).

As soon as a print context is created, the print attributes within can be accessed and modified by calling **`XpGetAttributes`** and **`XpSetAttributes`**, and the event selections within can be modified by calling **`XpSelectInput`** and **`XpInputSelected`**. Other Xp calls that explicitly take a print context id as a parameter will operate directly on that print context. All Xp and X calls without a print context id parameter (for example, all rendering oriented calls like **`XpStartJob`** and **`XDrawLine`**) require that a print context be set on the display connection (see **`XpSetContext`**). Failure to set a print context prior to calling a print-context-dependent call will result in the generation of an **`XPBadContext`** error.

A *`context_id`* for the print context is returned by **`XpCreateContext`**, and can be used to set the print context on display connections by calling **`XpSetContext`**. These display connections can be made from the same client (process) that called **`XpCreateContext`**, and from other clients (processes) that have acquired the *`context_id`* (for example, by inter-process communication). It is the responsibility of the clients sharing a print context to coordinate their usage of **`XpDestroyContext`** so that in-use print contexts are not prematurely destroyed.

The *`context_id`* remains valid for all clients until 1) the client creating the print context closes its *`display`* connection, or 2) any client calls **`XpDestroyContext`**. The *`context_id`* can be kept valid after the creating client's *`display`* connection closes if **`XSetCloseDownMode`** is called on *`display`* with `RetainPermanent` or `RetainTemporary`.

After creating a print context, and possibly modifying the `XPDocAttr` attribute `document-format` using a value from the list of available formats shown in the `XPPrinterAttr` attribute `document-formats-supported`, it is important to ask the *X Print Server* via **`XpGetScreenOfContext`** which screen has been associated with the print context, and create all X-resources that will be used in the print job on that screen. Failure to do so will result in undefined behavior. Once **`XpStartDoc`** is called, it is guaranteed that the screen returned by **`XpGetScreenOfContext`** will not change, hence this become the best time to call **`XpGetScreenOfContext`** and to start creating X-resources for the print job.

When **`XpCreateContext`** is called, the *client's* locale (see **`XpSetLocaleHinter`**) is included in the request as a “hint” to the *X Print Server*. If supported by the implementation, the *X Print Server* will use the hint to initialize the attribute pools with any localized attribute values (for example, the human

CDEnext

readable `XPPrinterAttr` attribute “descriptor” may be available in several different languages, and the hint will be used to select one). If the *X Print Server* cannot understand the hint, the *X Print Server* will typically fall back to the locale it is running in.

DEPENDENCIES

Code Sets

XpCreateContext, **XpGetAttributes**, **XpGetOneAttribute**, **XpSetAttributes**, **XpGetPrinterList**, **XpGetPdmStartParams** and **XpNotifyPdm** send and receive values expressed as COMPOUND_TEXT - a format for multiple character set data, such as multi-lingual text. Since the *X Print Server* may not be able to convert an arbitrary code set sent to its default character set, calls that require the *X Print Server* to interpret the value may fail (for example, **XpCreateContext** is called with a *printer_name* that is expressed in some unconvertable code set, so the *X Print Server* cannot compare it to the list of printer names it knows).

ERRORS/WARNINGS

This function can generate one of the following errors:

BadMatch	The specified *printer_name* does not exist on *display*. If the *context_id* is per chance used in other calls prior to receiving this error, those calls will fail with a **XPBadContext**.

SEE ALSO

XpSetContext, XpDestroyContext

CDEnext

2.3 XpSetContext

2.3.1 Short Description

Sets (i.e. associates) or unsets a print context with the specified display connection to the *X Print Server*.

2.3.2 Long Description

NAME

XpSetContext - sets or unsets a print context with a display connection

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpSetContext(
   Display          *display,
   XPContext        print_context)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
print_context	A pre-existing print context on the same X-Server.

RETURN VALUE

None.

DESCRIPTION

`XpSetContext` sets (i.e. associates) a print context previously created and initialized by **`XpCreateContext`** with a *display* connection. All subsequent print operations that do not explicitly take a print context id (for example, **`XpStartJob`**) on *display* will use and act upon the print context set by this call, until the print context is unset or **`XpDestroyContext`** is called. The print context can be set and used on subsequent jobs if not destroyed.

`XpSetContext`, if given `None` for *print_context*, will unset (disassociate) the print context previously associated with *display*. If there was no previously associated print context, no action is taken. The content of the formerly associated print context is not affected by this call, and other display connections may continue to use the print context.

Since font capabilities can vary from printer to printer (for example, PCL vs Postscript), **`XpSetContext`** may modify the list of available fonts (see **`XListFonts`**) on *display*, and the actual set of usable fonts (for example, see **`XLoadFont`**). A unique mix of true X fonts (see `fs(1)`), soft printer fonts masquerading as X fonts, and built-in printer fonts masquerading as X fonts may be available from within a given print context; a client should not assume all fonts available outside a print context will be available from within a print context. In other words, after calling **`XpSetContext`**, the global font path (see **`XGetFontPath`**) shows all "potential" sources of fonts, **`XListFonts`** shows all valid font names, but only **`XLoadFont`** will determine for sure if a specific instance of a font can be used.

Depending on the Print X-Server implementation, additional attributes may be available to help control which fonts can and can not be seen when a print context is set. For example, it may be desirable to show only the higher-performance built-in fonts when a print context is set, and ignore fonts that would have to be downloaded into the printer prior to printing.

When the print context is unset or **`XpDestroyContext`** is called, the font capabilities on *display* revert back to what they were previously.

See **`XpCreateContext`** for more details.

ERRORS/WARNINGS

This function can generate one of the following errors:

`XPBadContext`	The specified print context id is not valid.

SEE ALSO

`XpCreateContext, XpDestroyContext`

2.4 XpGetContext

2.4.1 Short Description

Get the current print context id for a display connection.

2.4.2 Long Description

NAME

XpGetContext - get the current print context id for a display connection

SYNOPSIS

```
#include <X11/extensions/Print.h>

XPContext XpGetContext(
   Display            *display)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.

RETURN VALUE

id of the current print context. `None` if there is no current print context.

DESCRIPTION

XpGetContext returns the *id* of the current print context associated with *display*. If a print context has not been set, a value of `None` is returned.

ERRORS/WARNINGS

This function can generate one of the following errors:

SEE ALSO

XpSetContext

2.5 XpDestroyContext

2.5.1 Short Description

Unsets and destroys a print context.

2.5.2 Long Description

NAME

XpDestroyContext - unset and destroy a print context.

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpDestroyContext(
   Display          *display,
   XPContext        print_context)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
print_context	A pre-existing print context.

RETURN VALUE

None.

DESCRIPTION

XpDestroyContext closes any outstanding associations between the print context and any display connections, and then destroys the print context to reclaim memory. All display connections using the print context will no longer be able to access the print context or re-set it (i.e. **XpSetContext**) in the future.

Since all X Print calls act upon a print context, prematurely destroying a print context will cause in-progress pages, documents and jobs to be canceled within the *X Print Server*.

ERRORS/WARNINGS

This function can generate one of the following errors:

XPBadContext	The specified print context id is not valid.

SEE ALSO

XpCreateContext

2.6 XpGetScreenOfContext

2.6.1 Short Description

Get the screen associated with the specified print context.

CDEnext

2.6.2 Long Description

NAME

XpGetScreenOfContext - get the screen associated with the current print context

SYNOPSIS

```
#include <X11/extensions/Print.h>

Screen *XpGetScreenOfContext(
   Display            *display,
   XPContext          print_context)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
print_context	A pre-existing print context.

RETURN VALUE

A pointer to the associated screen.

DESCRIPTION

`XpGetScreenOfContext` returns the screen that is associated with the print context. This call must be made after **`XpStartDoc`** (see **`XpCreateContext`**) to determine which specific screen X-resources must be created on.

ERRORS/WARNINGS

This function can generate one of the following errors:

`XPBadContext`	The specified print context id is not valid.

SEE ALSO

`XpCreateContext`

2.7 XpGetPageDimensions

2.7.1 Short Description

Get the page dimensions for the current printer settings.

2.7.2 Long Description

NAME

XpGetPageDimensions - get the page dimensions for the current printer settings

SYNOPSIS

```
#include <X11/extensions/Print.h>

Status XpGetPageDimensions(
   Display             *display,
   XPContext           print_context,
   unsigned short      *width,              /* return value */
   unsigned short      *height,             /* return value */
   XRectangle          *reproducible_area)  /* return value */
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
print_context	A pre-existing print context.
width	The pixel width of the page currently selected in the print context.
height	The pixel height of the page currently selected in the print context.
reproducible_area	The net reproducible area of the page currently selected in the print context, expressed in pixel offsets and dimension.

RETURN VALUE

Returns a *Status* of `0` on failure, or `1` on success..

DESCRIPTION

XpGetPageDimensions considers the medium currently selected in the print context (derived in part from default-medium, default-input-tray, input-trays-medium, content-orientation, default-resolution), and returns the total width and height of the page in pixels, and the net reproducible area within the total width and height.

ERRORS/WARNINGS

This function can generate one of the following errors:

XPBadContext	The specified print context id is not valid.

SEE ALSO

XpSetContext

2.8 XpStartJob - XpEndJob - XpCancelJob

2.8.1 Short Description

Indicates the beginning and ending of a single print job.

2.8.2 Long Description

NAME

XpStartJob - start a print job

XpEndJob - end a print job

XpCancelJob - cancel a print job

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpStartJob(
   Display              *display,
   XPSaveData           save_data)

void XpEndJob (
   Display              *display)

void XpCancelJob (
   Display              *display)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
save_data	If set to `XPSpool`, then the document data is typically sent to a spooler. If `XPGetData`, then the document data is made available to **`XpGetDocumentData`**.

RETURN VALUE

None.

DESCRIPTION

`XpStartJob` signals the beginning of a new print job.

If *`save_data`* is `XPGetData`, then the *X Print Server* buffers the document output for retrieval by **`XpGetDocumentData`**. The *X Print Server* will block the client's *`display`* connection if the output buffer grows too large, either because **`XpGetDocumentData`** has not been called, or **`XpGetDocumentData`** has been called but is not consuming document data fast enough.

CDEnext

The XPSaveData values for *save_data* are defined in <X11/extensions/Print.h>:

```
#define XPSpool        1    /* Job data sent to spooler */
#define XPGetData      2    /* Job data via XpGetDocumentData */
```

XpEndJob signals the end of a print job. All resulting job data is assembled and combined with data previously sent by **XpPutDocumentData**. If *save_data* is XPSpool, then the *ddx driver* within the *X Print Server* submits the results, typically to a print spooler (for example, *lp*).

XpCancelJob cancels an in-progress job. If *save_data* is XPGetData, then data available to **XpGetDocumentData** is no longer guaranteed to be valid or recoverable for the current job. If *save_data* is XPSpool, then the *X Print Server* will discard previously generated document data and reset in preparation for future jobs. Depending on the driver and spooler configuration, a partial document or page may be generated (e.g. the driver is directly attached to a device, and cannot recall data).

All changes to the XPJobAttr attribute pool (see **XpSetAttributes**) must be made prior to calling **XpStartJob**, after which a **XPBadSequence** will be generated if changes are attempted, until **XpEndJob** is called.

The last XPJobAttr attribute to be modified prior to the PrintStartJob protocol request is when **XpStartJob** internally calls **XpSetAttributes** on the job attribute job-owner. On POSIX systems, the job-owner attribute is set using getpwuid(3c) on the result of getuid(3c). The job-owner attribute can then be read by others (but not written) to determine job ownership, and may be used by the *X Print Server* to submit jobs to the spooler when *save_data* is XPSpool.

For clients selecting XPPrintMask (see **XpSelectInput**), the event XPPrintNotify will be generated with its detail field set to XPStartJobNotify or XPEndJobNotify when the *X Print Server* has completed **XpStartJob** and **XpEndJob** respectively.

XPEndJobNotify indicates when the document data has been sent to the spooler (*save_data*=XPSpool) or been completely sent to the client via **XpGetDocumentData** (*save_data*=XPGetData) - it does not mean that the client has received all the document data.

For more information, see documentation on the *Print Dialog Manager (PDM)* and *X Print Server Drivers*.

Conceptually, a "Job" is a collection of "Documents", where each Document is in turn a collection of "Pages". Depending on the print facilities underlying the *X Print Server* (for example, the *Distributed Print Management Facility (PDMF)*), these delineations may be mapped by a *ddx driver* into real functionality (e.g. see the server attribute multiple-documents-supported).

ERRORS/WARNINGS

These functions can generate one of the following errors:

XPBadContext	A valid print context id has not been set prior to making this call.
XPBadSequence	The function was not called in the proper order with respect to the other X Print Service Extension calls (example, **XpEndJob** prior to **XpStartJob**).
BadValue	The value specified for *save_data* is not valid.

SEE ALSO

XpSetContext, XpPutDocumentData, **XpSelectInput**, **XpSetAttributes**

2.9 XpStartDoc - XpEndDoc - XpCancelDoc

2.9.1 Short Description

Indicates the beginning and ending of a single print document within a print job.

2.9.2 Long Description

NAME

XpStartDoc - start a print document

XpEndDoc - end a print docoument

XpCancelDoc - cancel a print document

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpStartDoc (
   Display              *display,
   XPDocumentType       type)

void XpEndDoc (
   Display              *display)

void XpCancelDoc (
   Display              *display)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
type	Specifies the type of document: `XPDocRaw` or `XPDocNormal`

RETURN VALUE

None.

DESCRIPTION

XpStartDoc signals the beginning of a new print document.

If *type* is XPDocRaw, then the client will provide all the data for the resulting document using **XpPutDocumentData**; the *X Print Server* will not write any data into the resulting document. Calling **XpStartPage** in a XPDocRaw document will generate a **XPBadSequence** error. For more information, see **XpPutDocumentData**.

If *type* is XPDocNormal, then the *X Print Server* will generate document data, and depending on the *ddx driver*, can incorporate additional data from **XpPutDocumentData** into the output. For more information, see **XpPutDocumentData**.

The XPDocumentType values for *type* are defined in <X11/extensions/Print.h>:

```
#define XPDocNormal     1     /* Doc data handled by Xserver*/
#define XPDocRaw        2     /* Doc data passed through Xserver*/
```

XpEndDoc signals the end of a print document. All resulting document data is assembled and combined with data previously sent by **XpPutDocumentData**.

XpCancelDoc cancels an in-progress document. If *save_data* to **XpStartJob** is XPGetData, then data available to **XpGetDocumentData** is no longer guaranteed to be valid or recoverable for the current job. If *save_data* to **XpStartJob** is XPSpool, then the *X Print Server* will discard previously generated document data and reset in preparation for future documents. Depending on the driver and spooler configuration, a partial page or document may be generated (e.g. the driver is directly attached to a device, and cannot recall data).

Unlike **XpStartJob**, **XpEndJob**, **XpStartPage** and **XpEndPage**, an application is not required to call **XpStartDoc** and **XpEndDoc** in the process of printing. The “document” delineation may not be useful from the application’s or spooler’s perspective, hence is made optional. If **XpStartPage** is called immediately after **XpStartJob**, then a synthetic **XpStartDoc** with XPDocNormal will be generated by the *X Print Server* prior to **XpStartPage** (i.e., XPStartDocNotify and XPStartPageNotify will have the same sequence number). Likewise, if **XpEndJob** is called immediately after **XpEndPage**, then a synthetic **XpEndDoc** will be generated by the *X Print Server* prior to **XpEndJob** (i.e., XPEndDocNotify and XPEndJobNotify will have the same sequence number).

All changes to the XPDocAttr attribute pool (see **XpSetAttributes**) must be made prior to calling **XpStartDoc**, after which a **XPBadSequence** will be generated if changes are attempted, until **XpEndDoc** is called.

For clients selecting XPPrintMask (see **XpSelectInput**), the event XPPrintNotify will be generated with its detail field set to XPStartDocNotify or XPEndDocNotify when the *X Print Server* has completed **XpStartDoc** and **XpEndDoc** respectively.

XPStartDocNotify need not be received prior to calling **XpStartPage**.

For more information, see documentation on the *Print Dialog Manager (PDM)* and *X Print Server Drivers*.

Conceptually, a “Job” is a collection of “Documents”, where each Document is in turn a collection of “Pages”. Depending on the print facilities underlying the *X Print Server* (for example, the *Distributed Print Management Facility (PDMF)*), these delineations may be mapped by a *ddx driver* into real functionality (e.g. see the server attribute multiple-documents-supported).

ERRORS/WARNINGS

These functions can generate one of the following errors:

XPBadContext	A valid print context id has not been set prior to making this call.
XPBadSequence	The function was not called in the proper order with respect to the other X Print Service Extension calls (example, **XpEndDoc** prior to **XpStartDoc**).
BadValue	The value specified for *type* is not valid.

SEE ALSO

XpPutDocumentData, **XpSelectInput, XpSetAttributes**

2.10 XpPutDocumentData

2.10.1 Short Description

Allows an application to send and incorporate data into the output.

2.10.2 Long Description

NAME

XpPutDocumentData - send data to the output

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpPutDocumentData (
   Display              *display,
   Drawable             drawable,
   unsigned char        *data,
   unsigned long        data_len,
   char                 *doc_fmt,
   char                 *options)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
drawable	For `XPDocRaw` documents, must be `None`. For `XPDocNormal` documents, the destination drawable for rendering.
data	Specifies the device-specific data sent.
data_len	Specifies the number of bytes in *data*.
doc_fmt	Specifies the type of data sent. See below for valid values. String limited to XPCS characters.

options Specifies *ddx driver* dependent options. String limited to XPCS characters.

RETURN VALUE

None.

DESCRIPTION

Depending on *type* for **XpStartDoc**, XPDocRaw or XPDocNormal, **XpPutDocumentData** has two modes of operation.

In XPDocRaw mode, **XpPutDocumentData** sends *data* directly to the output, and *drawable* must be None, else a BadDrawable will be generated. The *X Print Server* does not emit document or page control codes into the output, and *data* is passed through unmodified. This is useful for sending previously constructed and complete documents (e.g. a PCL file), and using the *X Print Server*'s job control and submission capabilities. Since XPDocRaw implies that the client will be generating all document output, an XPBadSequence error will be generated if **XpStartPage** is called in this mode. The printer attribute xp-raw-formats-supported defines the valid values for *doc_fmt* in this mode, with unsupported values for *doc_fmt* causing BadValue to be generated.

In XPDocNormal mode, **XpPutDocumentData** sends data to the *X Print Server*, and depending on the *ddx driver* implementation, integrates *data* into the output. The parameters *doc_fmt* and *options* describe the format of *data* which guides the *ddx driver* in interpreting *data*. The printer attribute xp-embedded-formats-supported defines the valid values for *doc_fmt* in this mode, with unsupported values for *doc_fmt* causing BadValue to be generated.

Depending on the *ddx driver* implementation in use, **XpPutDocumentData** could be used, for example, to send a simple text file to a *Postscript ddx driver* that is capable of wrapping the appropriate document and page control constructs around the text so that it can be printed on a Postscript printer. Likewise, Encapsulated Postscript Files could be handled. Another use could be to send a TIFF file to a *PCL ddx driver* that can convert the image from TIFF into PCL and then integrate it into the current PCL output.

There is no limit to the size of *data* that can be sent to the server using **XpPutDocumentData**. **XpPutDocumentData** automatically decomposes the call into multiple protocol requests to make sure that the maximum request size of the server is not exceeded.

ERRORS/WARNINGS

The function can generate one of the following errors:

XPBadContext A valid print context id has not been set prior to making this call.

XPBadSequence The function was not called in the proper order with respect to the other X Print Service Extension calls (example, **XpPutDocumentData** prior to **XpStartDoc**).

BadValue The value specified for *doc_fmt* is not valid.

BadDrawable The value specified for *drawable* is not valid.

XPBadResourceID The value specified for *drawable* is not valid for the print context and print screen. The resource in question must be created in the proper print context and on the correct screen.

CDEnext

SEE ALSO

2.11 XpGetDocumentData

2.11.1 Short Description

Setup callbacks to retrieve document data from a print context.

2.11.2 Long Description

NAME

XpGetDocumentData - setup callbacks to retrieve document data from a print context.

SYNOPSIS

```
#include <X11/extensions/Print.h>

Status XpGetDocumentData(
   Display          *data_display,
   XPContext        context,
   XPSaveProc       save_proc,
   XPFinishProc     finish_proc,
   XPointer         client_data)
```


ARGUMENTS

data_display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
context	The print context from which document data is to be retrieved.
save_proc	A proc to be registered and called repeatedly to save blocks of document data.
finish_proc	A proc to be registered and called once when the print job has completed and all document data has been sent to *save_proc*.
client_data	Passed to *save_proc* and *finish_proc* when called.

RETURN VALUE

`NULL` if **XpGetDocumentData** encounters an error, non-`NULL` otherwise.

DESCRIPTION

XpGetDocumentData registers callbacks that allow a “consumer” to continuously retrieve document data generated in the *X Print Server* by a separate “producer”, where both are referencing the same print context. Though **XpGetDocumentData** retrieves document data, its lifetime is bounded by

XpStartJob and **XpEndJob**. **XpGetDocumentData** always returns immediately; if an error occurs and the callbacks cannot be registered, NULL is returned, otherwise a non-NULL is returned and the callbacks will be called sometime after the return from **XpGetDocumentData**. This producer/consumer model is initiated when **XpStartJob** is called by the producer with *save_data* equal XPGetData, and is subsequently enabled when **XpGetDocumentData** is called by the consumer.

Once **XpGetDocumentData** is called on *data_display*, *data_display* cannot be used for any additional X requests until *finish_proc* is called and returns. Further, *data_display* cannot be closed from within *save_proc* or *finish_proc*. To avoid deadlock, the producer and consumer must run in separate processes, or in separate threads of a single process.

After **XpGetDocumentData** successfully registers the callbacks (ie, it returns non-NULL), any generated X errors (for example, BadAlloc) or Xp errors (for example, XPBadContext or XPBadSequence) that are the result of **XpGetDocumentData** will cause the appropriate errors to be generated (see **XSetErrorHandler**), and will cause *finish_proc* to be called with a status of XPGetDocError. Any other activities (for example, a separate process destroying the print context) that prove fatal to the progress of **XpGetDocumentData** will also cause *finish_proc* to be called with a status of XPGetDocError.

If **XpGetDocumentData** is called prior to **XpStartJob**, then a **XPBadSequence** error is generated and *finish_proc* is called with XPGetDocError. If **XpGetDocumentData** is called after **XpStartJob** and *save_data* was specified as XPSpool, then a **XPBadSequence** error is generated and *finish_proc* is called with XPGetDocError. If the producer starts generating data, and a consumer has not been established or the consumer cannot consume data quickly enough, then the producer's display connection will be blocked by the *X Print Server.*

save_proc, as registered by **XpGetDocumentData**, is repeatedly called on each block of document data sent by the *X Print Server* until either **XpEndJob** or **XpCancelJob** is called. Each block provided to *data* will be *data_len* bytes in length, and the memory for *data* itself will be owned by **XpGetDocumentData**, so *save_proc* should copy *data* to another location before returning. After the last block of data has been delivered to *save_proc*, *finish_proc* is called with final status. The signatures for *save_proc* and *finish_proc* are defined in <X11/extensions/Print.h> as follows:

```
typedef void (*XPSaveProc)( Display *data_display,
                           XPContext context,
                           unsigned char *data,
                           unsigned long data_len,
                           XPointer client_data);

typedef void (*XPFinishProc)( Display *data_display,
                             XPContext context,
                             XPGetDocStatus status,
                             XPointer client_data);
```

Until **XpEndJob** or **XpCancelJob** is called, it is possible that various XPPrintNotify events will be generated (for example, a page has been canceled). It is left as an exercise to the consumer (or producer) whether to select for these events and terminate *save_proc* (by calling **XpEndJob** or **XpCancelJob**) in response to canceled pages or documents. Since data is considered opaque until the print job completes *successfully* (i.e. no cancellations), and there is no correlation between the sequence of data blocks and received events, and the consumer may not be able to interpret the byte stream, consumers may want to terminate *save_proc* upon receipt of cancellation events.

When `finish_proc` is called, sometime after **`XpGetDocumentData`** is called and returns, the following `status` codes as defined in <X11/extensions/Print.h> are returned:

```
#define XPGetDocFinished            0     /* normal termination */
#define XPGetDocSecondConsumer      1     /* setup error */
#define XPGetDocError               2     /* progress error */
```

`XPGetDocFinshed` (zero) indicates that all intended document data has been delivered by way of `save_proc`. All cancellation events are guaranteed to have arrived by the time `finished_proc` is called, and they should be taken into consideration for evaluating the validity of the document data returned.

`XPGetDocSecondConsumer` indicates that a consumer had already been established for the print context. The *X Print Server* only supports one consumer per print context.

`XPGetDocError` indicates that an error has been generated (for example, XPBadContext or XPBadSequence) and that no further document data will be delivered by the *X Print Server* to `save_proc`.

After `finish_proc` returns, both `save_proc` and `finish_proc` are unregistered and will no longer be called.

CDEnext

ERRORS/WARNINGS

`XPBadContext`	The specified print context id is not valid.
`XPBadSequence`	The function was not called in the proper order with respect to the other X Print Service Extension calls (example, **`XpGetDocumentData`** prior to **`XpStartJob`**).

SEE ALSO

`XpStartJob, XpDestroyContext`

2.12 XpStartPage - XpEndPage - XpCancelPage

2.12.1 Short Description

Indicates the beginning and ending of a single print page within a document.

2.12.2 Long Description

NAME

XpStartPage - start a print page

XpEndPage - end a print page

XpCancelPage - cancel a print page

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpStartPage (
   Display          *display,
   Window           window,
   Bool             exposures)

void XpEndPage (
   Display          *display)

void XpCancelPage (
   Display          *display)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
window	Specifies the window ID (a top level print window).
exposures	Boolean that specifies if exposure events should be generated per the below discussion.

RETURN VALUE

None.

DESCRIPTION

XpStartPage signals the beginning of a new print page, with *window* serving as the drawable representing the page. *window* is required to be a top level window, else a **BadWindow** is generated. No generation of document data will occur for rendering operations against *window* or its inferiors prior to **XpStartPage** or after **XpEndPage**.

XpStartPage causes *window* to be resized to the size of the media selected - the actual “printable area” may be smaller than the media size (see **XpGetPageDimensions**). Then **XpStartPage** causes the windows in the *window* hierarchy to be cleared to their background with the same constraints as **XClearArea(***display*, *window*, 0, 0, 0, 0, *exposures***)**, and exposure events to be generated for each window if *exposures* if `True` and the client has elected to receive `Expose` events using **XSelectInput**.

Within the **XpStartPage** and **XpEndPage** sequence, attempts to resize, move or unmap *window* will yield undefined results. Attempts to resize or move inferiors of *window* will be done with the same contraits as `ConfigureNotify`, except that the contents of any configured window may be lost; an `Expose` event will be generated if a window’s contents are lost.

XpEndPage signals the end of a print page. All resulting page data is assembled and combined with data previously sent by **XpPutDocumentData**.

CDEnext

XpCancelPage cancels an in-progress page. If *save_data* to **XpStartJob** is XPGetData, then data available to **XpGetDocumentData** is no longer guaranteed to be valid or recoverable for the current job. If *save_data* to **XpStartJob** is XPSpool, then the *X Print Server* will discard previously generated page data and reset in preparation for future pages. Depending on the driver and spooler configuration, a partial page may be generated (e.g. the driver is directly attached to a device, and cannot recall data).

All changes to the XPPageAttr attribute pool (see **XpSetAttributes**) must be made prior to calling **XpStartPage**, after which a **XPBadSequence** will be generated if changes are attempted, until **XpEndPage** is called.

For clients selecting XPPrintMask (see **XpSelectInput**), the event XPPrintNotify will be generated with its detail field set to XPStartPageNotify or XPEndPageNotify when the *X Print Server* has completed **XpStartPage** and **XpEndPage** respectively. If the event Expose is also selected for (see **XSelectInput**), the exposure events will be generated prior to XPPrintNotify.

XPStartPageNotify need not be received prior to calling any other X rendering routines.

For more information, see documentation on the *Print Dialog Manager (PDM)* and *X Print Server Drivers*.

Conceptually, a “Job” is a collection of “Documents”, where each Document is in turn a collection of “Pages”. Depending on the print facilities underlying the *X Print Server* (for example, the *Distributed Print Management Facility (PDMF)*), these delineations may be mapped by a *ddx driver* into real functionality (e.g. see the server attribute multiple-documents-supported).

CDEnext

ERRORS/WARNINGS

These functions can generate one of the following errors:

XPBadContext	A valid print context id has not been set prior to making this call.
XPBadSequence	The function was not called in the proper order with respect to the other X Print Service Extension calls (example, **XpEndPage** prior to **XpStartPage**).
BadWindow	The value specified for *window* is not valid.
XPBadResourceID	The value specified for *window* is not valid for the print context and print screen. The resource in question must be created in the proper print context and on the correct screen.
BadValue	The value specified for *exposures* is not valid.

SEE ALSO

XpPutDocumentData, **XpSelectInput, XpSetAttributes**

2.13 XpSelectInput

2.13.1 Short Description

Selects which X Print events from the specified print context the client is interested in.

2.13.2 Long Description

NAME

XpSelectInput - selects which X Print events from the specified print context the client is interested in.

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpSelectInput (
   Display            *display,
   XPContext          context,
   unsigned long      event_mask)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
context	The print context from which to select events.
event_mask	Specifies the event mask. This mask is the bitwise OR of one or more of the valid events mask bits (see below).

RETURN VALUE

None.

DESCRIPTION

XpSelectInput selects which X Print events from the specified print context the client is interest in. The X Print Events are generated from a current print context, and *not* from a window as is the case with **XSelectInput**.

The bits for *event_mask* are defined in `<X11/extensions/Print.h>`:

```
#define XPNoEventMask              0
#define XPPrintMask                (1L<<0)
#define XPAttributeMask            (1L<<1)
```

The resulting events are defined in `<X11/extensions/Print.h>`:

```
#define XPPrintNotify              0
#define XPAttributeNotify          1
```

ERRORS/WARNINGS

The function can generate one of the following errors:

`XPBadContext`	The specified print context id is not valid.
`BadValue`	The value specified for *`event_mask`* is not valid.

SEE ALSO

`XpInputSelected`

2.14 XpInputSelected

2.14.1 Short Description

Query which X Print events from the specific print context the client has selected to receive.

2.14.2 Long Description

NAME

XpInputSelected - query which X Print events from the specified print context the client has selected to receive

SYNOPSIS

```
#include <X11/extensions/Print.h>

unsigned long XpInputSelected (
   Display             *display,
   XPContext           context,
   unsigned long       *all_event_mask)  /* return value */
```

ARGUMENTS

`display`	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
`context`	The print context to which the query is being made.
`all_event_mask`	The set of events any client have selected.

RETURN VALUE

A *`event_mask`* bit mask describing which event classes the client has selected to receive.

DESCRIPTION

XpInputSelected queries which X Print events from the specified print context the client has selected to receive. The X Print Events are generated from a print context, and *not* from a window as is the case with **XSelectInput**. As events arrive, the *.context* field in the event can be used to determine which print context generated the event.

See **XpSelectInput** for the *event_mask* and *all_event_mask* values.

ERRORS/WARNINGS

The function can generate one of the following errors:

XPBadContext	The specified print context id is not valid.

SEE ALSO

XpSelectInput

2.15 XpGetAttributes

SHORT DESCRIPTION

Get an attribute pool from the specified print context.

2.15.1 Long Description

NAME

XpGetAttributes - get an attribute pool from the specified print context

SYNOPSIS

```
#include <X11/extensions/Print.h>

char *XpGetAttributes (
   Display          *display,
   XPContext        context,
   XPAttributes     type)
```

ARGUMENTS

display	Specifies a pointer to the Display structure; returned from **XOpenDisplay**.
context	Specifies the print context from which the attribute pool is to be retrieved.
type	Specifies the attribute pool.

RETURN VALUE

A `COMPOUND_TEXT` resource string *`pool`*, else `NULL` if any errors occured.

DESCRIPTION

`XpGetAttributes` returns *`pool`*, a `COMPOUND_TEXT` resource string representing the attribute pool specified by *`type`*. The caller is expected to free *`pool`* when it is no longer needed using `XFree(3)`.

The values for the typedef `XPAttributes` in `<X11/extensions/Print.h>` are:

```
#define XPJobAttr       1   /* get/set */
#define XPDocAttr       2   /* get/set */
#define XPPageAttr      3   /* get/set - subset of XPDocAttr */
#define XPPrinterAttr   4   /* get only (library) */
#define XPServerAttr    5   /* get only (library), no context needed */
```

The attribute pool (hence the resource string) consists of many name-value pairs (for example, `'copy-count: 3'`). The syntax of an attribute pool is the same as an X resource file (see “Resource Manager Functions, Section 15.1” in X Window System for X11R5 by Scheifler and Gettys).

Valid characters for each name (left hand side) are derived from the Posix Portable Filename Character Set (PPFCS), which is `"a"-"z"` and `"A"-"Z"` and `"0"-"9"` and `"_"` and `"-"`. Valid characters for each value (right hand side) are all characters except `NULL` and unescaped `NEWLINE`, though all predefined values in the X Print Service are confined to X Portable Character Set (XPCS) characters. Non XPCS values are typically limited to localized “description” strings. See **`XpCreateContext`** regarding the locale hint for more information on localized values.

ERRORS/WARNINGS

The function can generate of one of the following errors:

`XPBadContext`	The specified print context id is not valid.
`BadValue`	The value specified for *`type`* is not valid.
`BadAlloc`	Insufficient memory.

SEE ALSO

`XpCreateContext`

2.16 XpGetOneAttribute

SHORT DESCRIPTION

Get a single print attribute from the specified print context.

2.16.1 Long Description

NAME

XpGetOneAttribute - get a single print attribute from the specified print context.

SYNOPSIS

```
#include <X11/extensions/Print.h>

char *XpGetOneAttribute (
   Display             *display,
   XPContext           context,
   XPAttributes        type,
   char                *attribute_name)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
context	The print context from which the attribute value is being retrieved.
type	Type of pool from which the attribute will be retrieved.
attribute_name	Name of attribute to be returned.

RETURN VALUE

A COMPOUND_TEXT string *`attribute_value`*, else `NULL` if any errors occured.

DESCRIPTION

`XpGetOneAttribute` is a variation of **`XpGetAttributes`** to get a single attribute value from an attribute pool. Unlike the protocol used with **`XpGetAttributes`**, in which the reply contains an entire attribute pool, the protocol for **`XpGetOneAttribute`** has a reply with just one *`attribute_value`*.

When retrieving *`attribute_value`* (for example, from the name-value pair "`copy-count: 3`"), *`attribute_name`* should not include a colon (for example, "`copy-count`"), and the caller is expected to free the *`attribute_value`* returned (for example, "`3`") using `XFree(3)`.

`XpSetAttributes` can be used with the `XPAttrMerge` flag to set a single attribute.

ERRORS/WARNINGS

These functions can generate of one of the following errors:

`XPBadContext`	The specified print context id is not valid.
`BadValue`	The value specified for *`type`* is not valid.
`BadAlloc`	Insufficient memory.

CDEnext

SEE ALSO

XpSelectInput, XpSetContext, XrmGetStringDatabase

2.17 XpSetAttributes

SHORT DESCRIPTION

Set or update an attribute pool in the specified print context.

2.17.1 Long Description

NAME

XpSetAttributes - set or update an attribute pool in the specified print context

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpSetAttributes (
   Display            *display,
   XPContext          context,
   XPAttributes       type,
   char               *pool,
   XPAttrReplacement  replacement_rule)
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
context	The print context in which an attribute pool is to be set.
type	Specifies the type of the attribute pool to be set.
pool	An attribute pool represented as a resource string. String encoded in `COMPOUND_TEXT`.
replacement_rule	One of `XPAttrReplace` or `XPAttrMerge`.

RETURN VALUE

None.

DESCRIPTION

XpSetAttributes accepts *pool*, a `COMPOUND_TEXT` resource string representing new name-value pairs for the attribute pool specified by *type*. The attribute pool is modified by the new name-value pairs according to *replacement_rule*. For `XPAttrReplace`, the exisiting attribute pool is discarded and

CDEnext

replaced with *pool*. For XPAttrMerge, *pool* is merged into the existing attribute pool; pre-existing name-value pairs are replaced, and non-existing name-value pairs are added. The contents of *pool* is not affected by this call, and can be freed by the caller afterwords.

The values for the typedef XPAttributes in <X11/extensions/Print.h> are:

```
#define  XPJobAttr       1   /* get/set */
#define  XPDocAttr       2   /* get/set */
#define  XPPageAttr      3   /* get/set - subset of XPDocAttr */
#define  XPPrinterAttr   4   /* get only (library) */
#define  XPServerAttr    5   /* get only (library), no context needed */
```

The values for the typedef XPAttrReplacement in <X11/extensions/Print.h> are:

```
#define  XPAttrReplace   1
#define  XPAttrMerge     2
```

For attribute pools that are read-write (see “get/set” in XPAttributes definition), there exists in the XPPrinterAttr attribute pool an attribute for each and every read-write pool that shows all supported (settable) attributes. For example, in the XPPrinterAttr attribute pool can be found the attributes: job-attributes-supported, document-attributes-supported and xp-page-attributes-supported.

When setting supported attribute names, the X Print Server and associated driver will validate the new values and ignore those that are invalid - pre-existing values remain. When setting unsupported (i.e, unknown) attribute names, no validation is done, and the name-value pairs will be set, even though they will not be used. When deleting (i.e. failing to reset with XPAttrReplace) a supported attribute name, the X Print Server may respond by explicitly re-setting the attribute name with a default value, or may implicitly rely on an internal default without re-setting the attribute.

When setting certain supported attributes, the X Print Server and associated driver may choose to modify other associated attributes. For example, considering the XPPrinterAttr attribute document-formats-supported, setting the XPDocAttr attribute document-format may cause a number of other attributes to change.

For attribute pools that are read-only (see “get only” in XPAttributes definition), attempting to use **XpSetAttributes** generates a **BadMatch**. The *X Print Server* however is capable of modifying any attribute pool at any time, including those that are read-only for a client.

The lifetime of all attribute pools are bounded by the lifetime of the print context they are contained in. When set, all attribute values will “stick” across all Xp operations, until changed by the user directly, the *X Print Server* directly, or changed because of a side effect when either the user or *X Print Server* changed another attribute value.

For a complete description of all print attributes, the precedence between print attributes, and the “side effects” of setting certain print attributes on other print attributes, etc, See “X Print Service Attributes” on page 97.

CDEnext

To monitor changes to the attribute pools, see **XpSelectInput** and the event XPAttributeNotify. Since a print context can be shared among clients, changes made by one client will be seen by all others, and if selected for, the event XPAttributeNotify will be sent to all clients referencing the print context when changes do occur. It is the responsibility of the clients sharing a print context to coordinate their calls to **XpGetAttributes** and **XpSetAttributes** to avoid readers/writer problems.

ERRORS/WARNINGS

These functions can generate of one of the following errors:

XPBadContext	The specified print context id is not valid.
XPBadSequence	A request to set an attribute pool occured at a time when the attribute pool could not be modified (for example, modifying XPJobAttr immediately after calling **XpStartJob**).
BadValue	The value specified for *type* is invalid.
BadMatch	The attribute pool specified by *pool* cannot be set.
BadAlloc	Insufficient memory.

SEE ALSO

XpSelectInput, XpSetContext, XrmGetStringDatabase

2.18 XpGetPrinterList

SHORT DESCRIPTION

Retrieves a list of all printers supported on an *X Print Server.*

2.18.1 Long Description

NAME

XpGetPrinterList - retrieves a list of all printers supported on an *X Print Server*

SYNOPSIS

```
#include <X11/extensions/Print.h>

XPPrinterList XpGetPrinterList (
   Display            *display,
   char               *printer_name,
   int                *list_count) /* return value */
```

ARGUMENTS

display	Specifies a pointer to the print Display structure; returned from **XOpenDisplay**.

printer_name	Specifies the name of the printer for which information is desired. If NULL, then information is returned for all printers associated with the server.
list_count	The number of printers returned in *printer_list*.

RETURN VALUE

printer_list, or NULL if any errors occured.

DESCRIPTION

XpGetPrinterList returns a list of printer records where each record describes a printer supported by the *X Print Server*.

If *printer_name* is NULL, then a list of all printers supported is returned. If *printer_name* is non-NULL, only print records matching *printer_name* are returned, and if no records match *printer_name*, then NULL is returned.

printer_name takes a COMPOUND_TEXT string, and the *.name* and *.desc* fields in the returned list will be in COMPOUND_TEXT (note, ISO 8859-1 (Latin-1) is a proper subset of COMPOUND_TEXT, so can be used directly). If *printer_name* is in a code-set that the *X Print Server* cannot convert (into its operating code-set), then the *X Print Server* may fail to locate the requested printer. If *printer_name* is NULL, then all printer names, regardless of their code-set, can be returned, leaving the task of specific printer recognition up to the *caller*.

When **XpGetPrinterList** is called, the *client's* locale (see **XpSetLocaleHinter**) is included in the request as a "hint" to the *X Print Server*. If supported by the implementation, the *X Print Server* will use the hint to locate a localized description (*desc*) for each printer in the list. If the *X Print Server* cannot understand the hint, the *X Print Server* will typically fall back to descriptions in the locale it is running in.

Printer lists can be freed by calling **XpFreePrinterList**.

STRUCTURES

The XPPrinterList structure defined in <X11/extensions/Print.h> contains:

```
typedef struct {
      char *name;                 /* name */
      char *desc;                 /* localized description */
} XPPrinterRec, *XPPrinterList;
```

ERRORS/WARNINGS

This function can cause the generation of one of the following errors:

BadAlloc	Insufficient memory.

SEE ALSO

XpFreePrinterList

CDEnext

2.19 XpFreePrinterList

SHORT DESCRIPTION

A convenience routine to free a printer list.

2.19.1 Long Description

NAME

XpFreePrinterList - a convenience routine to free a printer list

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpFreePrinterList (
   XPPrinterList       printer_list)
```

ARGUMENTS

printer_list A list of printer records returned by **XpGetPrinterList**.

RETURN VALUE

None.

DESCRIPTION

XpFreePrinterList frees a list of printer records allocated and returned by **XpGetPrinterList**.

ERRORS/WARNINGS

None.

SEE ALSO

XpGetPrinterList

2.20 XpRehashPrinterList

SHORT DESCRIPTION

Cause an X-Server to recompute the list of available printers.

2.20.1 Long Description

NAME

XpRehashPrinterList - cause the X-Server to recompute its list of available printers

SYNOPSIS

```
#include <X11/extensions/Print.h>

void XpRehashPrinterList (
   Display            *display)
```

ARGUMENTS

display	Specifies a pointer to the print `Display` structure; returned from **`XOpenDisplay`**.

RETURN VALUE

None.

DESCRIPTION

`XpRehashPrinterList` causes the *X-Server* to recompute (update) its list of available printers, and update the attributes for the printers. The intended usage of this routine is in a special tool that a system administrator can run after changing the printer topology. General applications are encouraged to use this call sparingly if at all, and let the system administrator control printer topology updates.

Depending on the print facilities underlying the *X-Server*, the *X-Server* may be able to detect changes in the printer topology and dynamically update to reflect the changes, or may not be able to detect the changes and will have to be notified via **`XpRehashPrinterList`**.

In-progress print contexts will not be affected by **`XpRehashPrinterList`** as long as their printer destination remains valid.

ERRORS/WARNINGS

None.

SEE ALSO

`XpGetPrinterList`

2.21 XpQueryVersion

2.21.1 Short Description

Query an X-Server to determine if it supports the X Print Service Extension, and if it does, which version of the X Print Service Extension.

2.21.2 Long Description

NAME

XpQueryVersion - query an X-Server to determine if it supports the X Print Service Extension

SYNOPSIS

```
#include <X11/extensions/Print.h>

Status XpQueryVersion (
   Display          *display,
   short            *major_version, /* return value */
   short            *minor_version) /* return value */
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
major_version	The major version if the X Print Service Extension exists, else zero.
minor_version	The minor version if the X Print Service Extension exists, else zero.

RETURN VALUE

A non-zero status if the X Print Service Extension exists; otherwise a status of zero.

DESCRIPTION

`XpQueryVersion` determines if the X Print Service Extension is present. A non-zero **`Status`** is returned if the extension is supported, otherwise a zero **`Status`** is returned. Furthermore, the major and minor version numbers are returned to indicate the level of X Print Service Extension support.

The X Print Service Extension is "passively" initialized on the first call to any X Print Service function. There is no need to explicitly initialize the X Print Service Extension.

ERRORS

None.

SEE ALSO

`XListExtension`, **`XFreeExtensionList`**

2.22 XpQueryExtension

2.22.1 Short Description

Query an X-Server to determine if it supports the X Print Service Extension, and if it does, what the offsets are for associated events and errors.

2.22.2 Long Description

NAME

XpQueryVersion - query an X-Server to determine if it supports the X Print Service Extension

SYNOPSIS

```
#include <X11/extensions/Print.h>

Bool XpQueryExtension (
   Display              *display,
   int                  *event_base_return, /* return value */
   int                  *error_base_return) /* return value */
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **XOpenDisplay**.
event_base_return	The base value for X Print Service Extension events.
error_base_return	The base value for X Print Service Extension errors.

RETURN VALUE

A non-zero status if the X Print Service Extension exists; otherwise a status of zero.

DESCRIPTION

XpQueryExtension determines if the X Print Service Extension is present. A non-zero value (True) is returned if the extension is supported, otherwise a zero value (False) is returned. If the extension is present, the base values for events and errors are returned, and can be used to decode incoming event and error values.

The X Print Service Extension is “passively” initialized on the first call to any X Print Service function. There is no need to explicitly initialize the X Print Service Extension.

ERRORS

None.

SEE ALSO

XListExtension, **XFreeExtensionList**

CDEnext

2.23 XpQueryScreens

2.23.1 Short Description

Query an X-Server to determine which of all the screens on an X Server support the X Print Service Extension.

2.23.2 Long Description

NAME

XpQueryScreens - query an X-Server to determine which of all the screens on an X Server support the X Print Service Extension

SYNOPSIS

```
#include <X11/extensions/Print.h>

Screen **XpQueryScreens (
   Display              *display,
   int                  *list_count)     /* return value */
```

ARGUMENTS

display	Specifies a pointer to the `Display` structure; returned from **`XOpenDisplay`**.
list_count	Specifies the number of screens contained in the return value.

RETURN VALUE

A non-`NULL` pointer of a list of screen pointers if one or more screens support the X Print Service Extension; otherwise a status of `NULL`.

DESCRIPTION

`XpQueryScreens` determines if the X Print Service Extension is present, and if so, which of all the screens on the X Server support the X Print Service Extension. Unlike many other extensions, the X Print Service Extension may be restricted to a subset of all available screens - for example, a single X-Server may be supporting video displays on some screens and printers on others.

The list of screens can be freed by calling `XFree(3C)`.

ERRORS

None.

CDEnext

SEE ALSO

2.24 XpGetPdmStartParams

2.24.1 Short Description

A standard convenience function to build up parameters in accordance with the PDM Selection Protocol

2.24.2 Long Description

NAME

XpGetPdmStartParams - build up parameters in accordance with the PDM Selection Protocol

SYNOPSIS

```
#include <X11/extensions/Print.h>
#include <X11/XpPdm.h>

Status XpGetPdmStartParams (
   Display          *print_display,
   Window           print_window,
   XPContext        print_context,
   Display          *video_display,
   Window           video_window,
   Display          **selection_display,   /* return value */
   Atom             *selection,            /* return value */
   Atom             *type,                 /* return value */
   int              *format,               /* return value */
   unsigned char    **data,                /* return value */
   int              *nelements)            /* return value */
```

ARGUMENTS

print_display	Specifies a pointer to the print `Display` structure; returned from **`XOpenDisplay`** on the *X Print Server*.
print_window	Specifies a client window on any screen of *print_display* long-lived enough for ICCCM communications of the final *PDM* status (“OK” or “CANCEL” `ClientMessage`) sent to *print_window*.
print_context	An existing print context that the *PDM* should reference.
video_display	Specifies a pointer to the video `Display` structure; returned from **`XOpenDisplay`** on the *Video X-Server*.
video_window	Specifies the window on *video_display* off which the transient dialogs from the *PDM* should be posted.

CDEnext

selection_display A returned display connection against which the PDM selection should be made. May be equal to *print_display* or *video_display*, or may be a new display connection that the caller should close when done.

selection A returned selection atom against which a PDM selection should be made.

type A returned type for the PDM Selection Protocol property the caller is expected to create.

format A returned format for the PDM Selection Protocol property the caller is expected to create.

data A returned data set for the PDM Selection Protocol property the caller is expected to create. The caller is expected to `XFree(3C)` the data when finished.

nelements A returned number of elements for the PDM Selection Protocol property the caller is expected to create.

RETURN VALUE

NULL status if an error occured, non-NULL otherwise.

DESCRIPTION

XpGetPdmStartParams is a convenience routine used to construct the necessary property information and selection display connection information needed to initiate a *PDM* Selection per the "PDM Selection Protocol". Once the information is constructed, the caller can manage the creation of a property, the generation of a `SelectionRequest`, the receipt of a `SelectionNotify` event, and the recept of a `ClientMessage` event, as described in the PDM Selection Protocol.

When finished, the caller is expected to free *data* using `XFree(3C)`.

If `NULL` is returned by **XpGetPdmStartParams**, then an error occured and all return values are invalid.

XpGetPdmStartParams also implements the concept of an "alternate selection server". By changing the name of the x-selection or by changing the *X-Server* upon which the selection is made, an alternate *PDM* can be engaged. By default, the *PDM* owning the `PDM_MANAGER` x-selection on the *Print X-Server* is the *PDM* engaged.

Setting the environment variable XPDMSELECTION causes **XpGetPdmStartParams** to use an alternate selection name. If not set, the selection name `PDM_MANAGER` is used.

Setting the environment variable XPDMDISPLAY causes **XpGetPdmStartParams** to locate the selection on an alternate *X Server*. If not set, *selection_display* (a returned parameter) is set equal to *print_display*. If set to one of the keywords "`print`" or "`video`", *selection_display* is set to *print_display* or *video_display*, respectively. If set to a valid `DISPLAY`-style string, *selection_display* may be set, as appropriate, to one of *print_display*, *video_display*, or to a *new* display connection made from within **XpGetPdmStartParams**. Only in the single case where a *new* display connection is made should the caller close *selection_display* using `XCloseDisplay(3)`.

When **XpGetPdmStartParams** is called, the *client's* locale (see **XpSetLocaleHinter**) is included in the returned information as a "hint" to the *Print Dialog Manager (PDM)*. If supported by the implementation, the *PDM* will use the hint to display dialogs more appropriately labeled for the locale of the client. If the *Print Dialog Manager* cannot understand the hint, the *PDM* will typically fall back to the locale it is running in. Note that the locale of the print attributes that the *PDM* will subsequently access have already been determined when the client called **XpCreateContext**.

ENVIRONMENT VARIABLES

See the above discussion regarding XPDMDISPLAY and XPDMSELECTION.

All environment variables are re-read each time **XpGetPdmStartParams** is called.

ERRORS/WARNINGS

SEE ALSO

2.25 XpGetAuthParams

2.25.1 Short Description

A standard convenience function to build up parameters in accordance with the PDM Selection Protocol

2.25.2 Long Description

NAME

XpGetAuthParams - build up parameters in accordance with the PDM Selection Protocol

SYNOPSIS

```
#include <X11/extensions/Print.h>

Status XpGetAuthParams (
   Display          *print_display,
   Display          *video_display,
   Display          **selection_display,   /* return value */
   Atom             *selection,            /* return value */
   Atom             *target)               /* return value */
```

ARGUMENTS

print_display	Specifies a pointer to the print `Display` structure; returned from **XOpenDisplay** on the *X Print Server*.
video_display	Specifies a pointer to the video `Display` structure; returned from **XOpenDisplay** on the *Video X-Server*.

CDEnext

selection_display	A returned display connection against which the PDM selection should be made. May be equal to *print_display* or *video_display*, or may be a new display connection that the caller should close when done.
selection	A returned selection atom against which the conversion should be made.
target	A returned target atom against which the conversion should be made.

RETURN VALUE

NULL status if an error occured, non-NULL otherwise.

DESCRIPTION

XpGetAuthParams is a convenience routine used to construct the necessary property information and selection display connection information needed to initiate a transfer of display-connection authorization information per the "PDM Selection Protocol". Once the information is constructed, the caller can manage the creation of a property, the generation of a `SelectionRequest`, the receipt of a `SelectionNotify` event, and the subsequent transmission of authorization information. See **XpSendAuth** and **XpSendOneTicket** for more details.

If `NULL` is returned by **XpGetAuthParams**, then an error occured and all return values are invalid.

XpGetAuthParams also implements the concept of an "alternate selection server". By changing the name of the x-selection or by changing the *X-Server* upon which the selection is made, an alternate *PDM* can be engaged. By default, the *PDM* owning the `PDM_MANAGER` x-selection on the *Print X-Server* is the *PDM* engaged.

Setting the environment variable XPDMSELECTION causes **XpGetAuthParams** to use an alternate selection name. If not set, the selection name `PDM_MANAGER` is used.

Setting the environment variable XPDMDISPLAY causes **XpGetAuthParams** to locate the selection on an alternate *X Server*. If not set, *selection_display* (a returned parameter) is set equal to *print_display*. If set to one of the keywords `"print"` or `"video"`, *selection_display* is set to *print_display* or *video_display*, respectively. If set to a valid `DISPLAY`-style string, *selection_display* may be set, as appropriate, to one of *print_display*, *video_display*, or to a *new* display connection made from within **XpGetAuthParams**. Only in the single case where a *new* display connection is made should the caller close *selection_display* using `XCloseDisplay(3)`.

ENVIRONMENT VARIABLES

See the above discussion regarding XPDMDISPLAY and XPDMSELECTION.

All environment variables are re-read each time **XpGetAuthParams** is called.

ERRORS/WARNINGS

SEE ALSO

2.26 XpSendAuth

2.26.1 Short Description

A standard convenience function to select and send display-connection authorization information in accordance with the PDM Selection Protocol

2.26.2 Long Description

NAME

XpSendAuth - select and send display-connection authorization information in accordance with the PDM Selection Protocol

SYNOPSIS

```
#include <X11/extensions/Print.h>

Status XpSendAuth (
   Display           *display,
   Window            window)
```

ARGUMENTS

display	Specifies a pointer to the print `Display` structure; returned from **`XOpenDisplay`** on the *Selection Print Server.*
window	Specifies the “mailbox window” as described in the PDM Selection Protocol for the PDM_MBOX target.

RETURN VALUE

NULL status if an error occured, non-NULL otherwise.

DESCRIPTION

`XpSendAuth` is a convenience routine that determines what display-connection authorization information needs to be sent to a *PDM*, and sends it. Programmers can write their own code to figure out what information needs to be sent, and then call **`XpSendOneTicket`**, or can simply call **`XpSendAuth`** and let Xp do the right thing. See **`XpGetAuthParams`** for the necessary setup details.

`XpSendAuth` considers authorization information in a .Xauthority style file pointed to by the environment variable XPDMXAUTHORITY. Any and all authorization information in the file may be sent to the *PDM* by the time **`XpSendAuth`** returns. Users can set XPDMXAUTHORITY to be the same

CDEnext

as the XAUTHORITY environment variable, or may wish to manage a separate subset authority file for printing purposes. If XPDMXAUTHORITY is not set, **XpSendAuth** does nothing and returns immediately.

If NULL is returned by **XpSendAuth**, then an error occured.

ENVIRONMENT VARIABLES

ERRORS/WARNINGS

SEE ALSO

2.27 XpSendOneTicket

2.27.1 Short Description

A standard convenience function to send one authorization ticket in accordance with the PDM Selection Protocol

2.27.2 Long Description

NAME

XpSendOneTicket - send one authorization ticket in accordance with the PDM Selection Protocol

SYNOPSIS

```
#include <X11/extensions/Print.h>
#include <X11/Xauth.h>

Status XpSendOneTicket (
   Display             *display,
   Window              window,
   Xauth               ticket,
   Bool                more)
```

ARGUMENTS

display	Specifies a pointer to the print Display structure; returned from **XOpenDisplay** on the *Selection Print Server.*
window	Specifies the “mailbox window” as described in the PDM Selection Protocol for the PDM_MBOX target.
ticket	The ticket to be sent. If NULL, a null ticket is sent.
more	A boolean indicating if more tickets will follow (True), or if this is the last ticket to be sent (False).

CDEnext

RETURN VALUE

NULL status if an error occured, non-NULL otherwise.

DESCRIPTION

`XpSendOneTicket` is a convenience routine used to send one display-connection authorization ticket to the *PDM* per the "PDM Selection Protocol". The ticket is decomposed as necessary, and `ClientMessages` are generated. See **`XpGetAuthParams`** for more details.

If `NULL` is returned by **`XpSendOneTicket`**, then an error occured.

ENVIRONMENT VARIABLES

ERRORS/WARNINGS

SEE ALSO

2.28 XpSetLocaleHinter & XpGetLocaleHinter

2.28.1 Short Description

Set and get the "locale hinter" function used by several Xp calls.

2.28.2 Long Description

NAME

XpSetLocaleHinter - set a "locale hinter" function and description of it

XpGetLocaleHinter - get a pointer to and description of the current "locale hinter" function

SYNOPSIS

```
#include <X11/extensions/Print.h>

XpSetLocaleHinter(
   XPHinterProc        hinter_proc,
   char                *hinter_desc)

char *XpGetLocaleHinter(
   XPHinterProc        *hinter_proc)
```

ARGUMENTS

hinter_proc	A pointer to a "hinter proc".
hinter_desc	A pointer to contextual information about the locale hinter proc.

RETURN VALUE

(XpSetLocaleHinter) None.

(XpGetLocaleHinter) The contextual information for the currently installed hinter.

DESCRIPTION

Since (to date) there is no single industry standard for locale values, locale information about the current client required by **XpCreateContext**, **XpGetPrinterList** and **XpGetPdmStartParams** is at best considered a “hint” when transmitted to the X Print Server and PDM. In single vendor environments, the locale hint should be consistent and understood. In multi-vendor environments however, the locale hint may or may not be understood. The caller locale will be used as the fallback default.

XpSetLocaleHinter and **XpGetLocaleHinter** access hooks that are used register more advanced hint generators. By default, Xp uses a hinter proc that calls `setlocale(3C)` on the `CTYPE` category on POSIX systems, and *hinter_desc* is `NULL`.

XpSetLocaleHinter sets the *hinter_proc* and *hinter_desc* which will be subsequently used by the Xp calls requiring a locale hint (see above). *hinter_proc* is the function that will generate the locale hint (for example, “C”), and *hinter_desc* is a string, with or without the embeddable keyword `%locale%`, that provides a higher level context for the results of *hinter_proc*.

If *hinter_proc* is set to `NULL`, then the default Xp hinter proc is installed. **XpSetLocaleHinter** makes its own private copy of *hinter_desc* prior to returning.

An example set call might look as follows:

```
XpSetLocaleHinter( my_hinter, “%locale%;CDElocale” );
```

Where *my_hinter* might look as follows:

```
char *my_hinter()
{
    /*
     * Use setlocale() to retrieve the current locale.
     */
    return( my_x_strdup( setlocale(LC_CTYPE, (char *) NULL) ) );
}
```

The signature for *hinter_proc* is defined in `<X11/extensions/Print.h>` as follows:

```
typedef char * (*XPHinterProc)();
```

hinter_proc is expected to return a string that can be freed using `XFree(3C)` by the Xp calls themselves.

XpGetLocaleHinter gets the currently installed *hinter_proc* and *hinter_desc*. The caller is expected to `XFree(3C)` the returned *hinter_desc*.

If both *hinter_desc* and the results of *hinter_proc* are non-`NULL`, and the keyword `%locale%` is found in *hinter_desc*, then the keyword will be replaced with the result of *hinter_proc*. The resulting string will be used as the locale hint by the Xp calls.

If both `hinter_desc` and the results of `hinter_proc` are non-`NULL`, but the keyword `%locale%` is not found in `hinter_desc`, then `hinter_desc`, as is, becomes the string used as the locale hint by the Xp calls.

If one of `hinter_desc` or the results of `hinter_proc` is NULL, then the other non-NULL value becomes the string used as the locale hint by the Xp calls.

If `hinter_desc` and the results of `hinter_proc` are NULL, then a `NULL` (i.e. `(char *) NULL`) locale hint is sent by the Xp calls.

The syntax for `hinter_desc` is a variation of the unadopted X/Open standard for a "String Network Locale-Specification Syntax" (X/Open, Distributed Internationalization Services, Version 2, 1994 Snapshot). The Xp `hinter_desc` syntax is:

```
name_spec[;registry_spec[;ver_spec[;encoding_spec]]]
```

Some examples include (`hinter_desc` to left, expanded results to the right):

```
CFRENCH                           CFRENCH
%locale%                          C
%locale%;CDElocale                C;CDElocale
%locale%;HP                       C;HP
%locale%;IBM                      C;IBM
%locale%;XOPEN;01_11;XFN-001001   de_DE;XOPEN;01_11;XFN-001001
```

In Xp, the first item is the locale name, followed by progressively more detailed information about the locale name, with each piece of information separated by a ';'.

CDEnext

ERRORS/WARNINGS

None

SEE ALSO

`XpCreateContext, XpGetPrinterList, XpGetPdmStartParams, XpNotifyPdm`

PDM SELECTION PROTOCOL

3.1 Overview

3.1.1 Short Description

The "PDM Selection Protocol" is based on ICCCM selections, and allows a *client* to transmit information and request that a *PDM* be started against a *Print X-Server* and *Video X-Server* per the information transmitted.

3.1.2 Long Description

For the sake of the following protocol discussion, the term *"Selection X-Server"* will be used to denote the X-Server on which the selection window, selection atom, type atoms and properties exist, and to which the *client* and *PDM* have active display connections for the purposes of communication. The term *"Print X-Server"* will be used to denote the X-Server on which printing will take place, and print attributes will be modified. In most cases, the *Selection X-Server* will be the same as the *Print X-Server*. The term *"Video X-Server"* will be used to denote the X-Server on which the *client* and *PDM* post dialogs to the user.

The *PDM* is a long lived process that establishes ownership of a well known selection, and responds to conversions on several target types requested by *client*s on the selection. The *PDM's* primary purpose is to provide dialogs that allow the user to configure the print options that the *client* will subsequently use in a printing job.

The *client* is a process that requests conversions on the well known selection, and its targets, in order to enlist the help of the *PDM*.

3.2 Setup of the Protocol

Specifically, the *PDM* establishes ownership of the well-known `PDM_MANAGER` selection by making the following `SetSelectionOwner` request:

`selection`	`PDM_MANAGER`
`owner`	a window of the *PDM*
`time`	something other than `CurrentTime`

Then the *PDM* establishes the capability to handle the following selection targets:

`PDM_START`	Used to request that a *PDM* be started.
`PDM_MBOX`	Used to request that a "mailbox window" be provided, so that *Video X-Server* connection authorization information (e.g. magic cookies) can be subsequently mailed by the *client* using ClientMessage's.
`TARGETS`	Used to query the list of supported targets.
`MULTIPLE`	Standard ICCCM capability.

TIMESTAMP	Used to query the timestamp that the *PDM* used to acquire ownership of the PDM_MANAGER selection.

The *client* can initiate a conversion on any of the supported targets. The supported targets are described below.

3.3 PDM_START Selection Target

The running of a *PDM* requires two phases to complete.

Phase 1 involves a SelectionRequest and SelectionNotify on the PDM_START target to start a *PDM*. During the SelectionRequest, *Video X-Server* and *Print X-Server* connection information is passed to the *PDM* in a *client*-owned property. During the SelectionNotify, *PDM* startup status is returned in the same property. For phase 1, the selection window, selection atom, all target atoms, type atoms, property atoms and property windows are created relative to the *Selection X-Server.*

Phase 2 involves a ClientMessage to the *client* carrying shutdown information about the *PDM*, typically "OK" or "CANCEL". For phase 2, all atom values and communication windows are created relative to the *Print X-Server.*

Phase 1 starts when the *client* creates the following:

requestor	A *client* owned window that will act as the selection requestor, and will hold *property*. This window can be the same as *client_window*. Note: when using with the PDM_MBOX target, it is important to use the same *requestor* window id.
property	A *client* owned property for the two-way transfer of information as described in *Phase 1* of the protocol.
client_window	A *client* owned window that will live long enough to handle the ClientMessage describing "OK" or "CANCEL" as described in *Phase 2* of the protocol.

Clients wishing to start a *PDM* can then make the following ConvertSelection request:

display	*selection_display* (may be the same as *print_display)*
selection	PDM_MANAGER
target	PDM_START
property	*property*
requestor	*requestor*
time	timestamp of the gesture that initiated the ConvertSelection request

The PDM_START target is parameterized, and the property named in the request (*property*) contains the following list of information:

video_display	representing the display in the form "host:display[.screen]"
video_window	representing the window id in the form "0x123"

print_display	representing the display in the form "host:display[.screen]"
print_window	representing the window id in the form "0x456"
print_context	representing the context id in the form "0x789"
locale	representing the locale in the form "C"

XmbTextListToTextProperty with an encoding style of XStdICCTextStyle and a list count of 6 can be used to generate *type* (from *tp.encoding*), *format* (*tp.format*), *mode* (PropModeReplace), *data* (*tp.value*) and *nelements* (*tp.nitems*) for *property* from the above list.

The PDM_START target has side effects. The *PDM* uses *property* to inform *client* of the success or failure of the PDM_START request. Specifically, *property* is created with *type* (XA_ATOM), *format* (32), *mode* (PropModeReplace), *data* (XInternAtom of below) and *nelements* (1).

The following are the name strings for the atom values used for *data*:

PDM_START_OK	The *PDM* was successfully started.
PDM_START_VXAUTH	The *PDM* was not authorized to connect to the *Video X-Server* specified.
PDM_START_PXAUTH	The *PDM* was not authorized to connect to the *Print X-Server* specified. This can happen if 1) the *Print X-Server* is different than the *Selection X-Server*, or 2) the *Print* and *Selection X-Server* are the same, and the authorization has been withdrawn since the time that the *PDM* made its initial connection to the *Print X-Server*.
PDM_START_ERROR	The *PDM* encountered an error. If the *PDM* supports logging of error messages, an error message could have been logged.

Finally, *client* should destroy *property* to indicate a successful ConvertSelection request.

Phase 1 of the *PDM* Selection Protocol is now complete. *Phase 2* is as follows.

When the *PDM* completes, it informs the *client* of the user's choice of "OK" or "CANCEL" via a ClientMessage. The message type of the ClientMessage is PDM_REPLY, the format is "32" (XA_ATOM), and *xclient.data.l[0]* contains an atom value.

The following are the name strings for the atom values used for *xclient.data.l[0]*:

PDM_EXIT_OK	The user's choice was "OK". The *PDM* may or may not have changed any printer attributes.
PDM_EXIT_CANCEL	The user's choice was "CANCEL". The *PDM* may have momentarily changed the printer attributes, but is now claiming to have restored them to their pre-convert-selection state.
PDM_EXIT_VXAUTH	The *PDM* was not authorized to connect to the *Video X-Server* specified.
PDM_EXIT_PXAUTH	The *PDM* was not authorized to connect to the *Print X-Server* specified.

CDEnext

`PDM_EXIT_ERROR`	The *PDM* encountered an error. If the *PDM* supports logging of error messages, an error message could have been logged. This return code *should only be used if* the *PDM* cannot restore the printer attributes and return a synthetic `PDM_EXIT_CANCEL`.

3.4 PDM_MBOX Selection Target

Clients hoping to start a *PDM* using the `PDM_START` target may have to first transfer display authorization information to the *PDM*, as the *PDM* will have to establish a display connection on the *Video X-Server*. Use of this target requires two phases to complete.

Phase 1 consists of a `SelectionRequest` and `SelectionNotify` on the `PDM_MBOX` target, and results in the *PDM* allocating a "mailbox window" for the *client*. *Phase 2* consists of the *client* using multiple `ClientMessages` to "mail" display authorization information (see `Xauth(1)`) to the mailbox window, thus transfering to the *PDM* authorization information that it may need to establish a connection to the *Video X-Server*.

Phase 1 starts when the *client* creates the following:

`requestor`	A *client* owned window that will act as the selection requestor, and will hold *`property`*. This window can be the same as *`client_window`* (see `PDM_START` target). Note: when using with the `PDM_START` target, it is important to use the same *`requestor`* window id.
`property`	A *client* owned property for the return of information as described later.

Clients wishing to acquire a "mailbox window" from the *PDM* can then make the following `ConvertSelection` request:

`display`	*`selection_display`*	(may be the same as *`print_display`*)
`selection`	`PDM_MANAGER`	
`target`	`PDM_MBOX`	
`property`	*`property`*	
`requestor`	*`requestor`*	
`time`	timestamp of the gesture that initiated the `ConvertSelection` request	

The `PDM_MBOX` target has side effects. The *PDM* creates a mailbox window on the *Selection X-Server* that the *client* can later use to mail (via ClientMessages) display authorization information. The *PDM* places the mailbox window id in *`property`*. Specifically, *`property`* is created with *`type`* (`XA_WINDOW`), *`format`* (`32`), *`mode`* (`PropModeReplace`), *`data`* (Window id) and *`nelements`* (`1`).

Finally, the *client* should destroy *`property`* to indicate a successful `ConvertSelection` request.

Phase 1 is now complete, and a mailbox window has been acquired. *Phase 2* is as follows.

CDEnext

The client can now send one or more display authorization “tickets” contained in `Xauth` structures (see `XauReadAuth(1)`) - tickets that the *PDM* can use to establish a connection to the *Video X-Server*. For each ticket, a “Ticket Header” `ClientMessage` is sent followed by one or more “Ticket Content” `ClientMessages`. A status flag in the Ticket Header indicates when the transfer of tickets has been or will be completed (e.g. this is the last ticket).

Note: which tickets need to be transfered is left as an exercise to the *client*. Savvy *clients* would use calls such as `XauGetBestAuthByAddr(1)` and ideally transfer just the one ticket needed so the *PDM* can connect to the *Video X-Server*. More likely, *clients* will use calls like `XauReadAuth(1)` to transfer one-by-one all the possible tickets, and then let the *PDM* figure out which one is really needed.

The `ClientMessage` for the “Ticket Header” is as follows:

type	`ClientMessage`
display	*selection_display*
window	mailbox window id from `ConvertSelection`
message_type	atom value for the string “`PDM_MAIL`”
format	`16`, which among other things identifies this `ClientMessage` as a Ticket Header message.
data.s[0]	`0` if this is a `NULL` and terminating ticket, `1` if this is a `non-NULL` and terminating (last) ticket, or `2` if this is a `non-NULL` ticket and more tickets will follow.
data.s[1]	ticket *address_length* from `Xauth` structure.
data.s[2]	ticket *number_length*
data.s[3]	ticket *name_length*
data.s[4]	ticket *data_length*
data.s[5]	ticket *family*

The `ClientMessage` for the “Ticket Contents” is as follows:

type	`ClientMessage`
display	*selection_display*
window	mailbox window id from `ConvertSelection`
message_type	atom value for the string “`PDM_MAIL`”
format	`8`, which among other things identifies this `ClientMessage` as a Ticket Content message.
data.b	the next block of char* data at most 20 bytes in length derived from and representing the concatination of the ticket’s *address* (from `Xauth` structure), *number*, *name* and *data*.

Since a `ClientMessage` with *`format`* equal `8` can transfer at most `20` bytes, the concatinated data for *`address`*, *`number`*, *`name`* and *`data`* must be incrementally sent over. For example, if *`address_length`*, *`number_length`*, *`name_length`* and *`data_length`* equals `39`, then a 20-byte ticket content message would be sent followed by a 19-byte message.

The transfer of tickets will not be considered complete until a NULL or last ticket has been sent.

3.5 TARGETS Selection Target

Returns a list of atoms that represent the targets for which an attempt to convert the `PDM_MANAGER` selection will succeed (barring unforseeable problems such as `Alloc` errors). This list will include all required atoms.

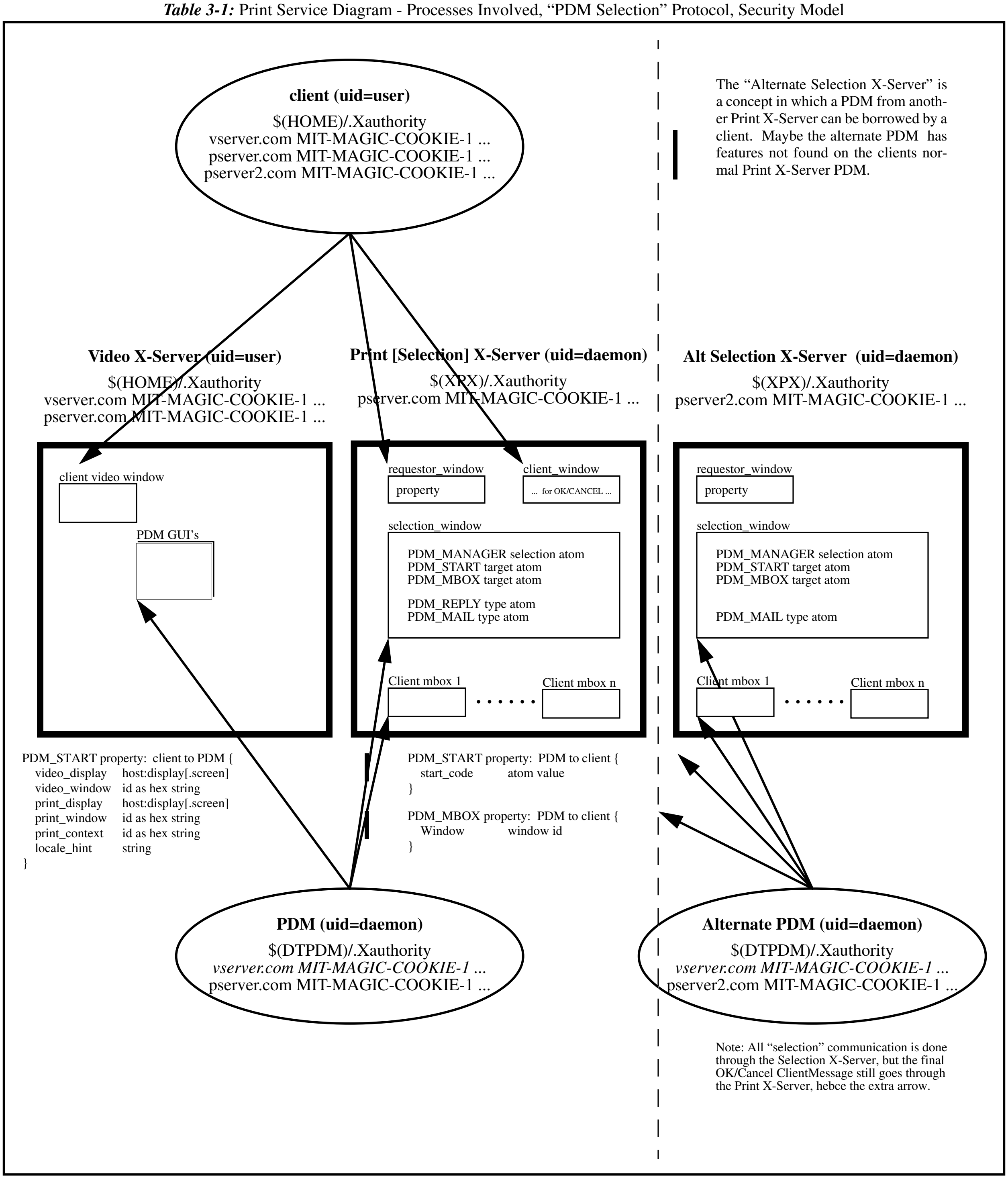
3.6 MULTIPLE Selection Target

Implements a sequence of `SelectionRequest` events to be ended with a single `SelectionNotify` event.

3.7 TIMESTAMP Selection Target

Returns the timestamp that the owner of the PDM_MANAGER selection used to acquire the selection.

Table 3-1: Print Service Diagram - Processes Involved, "PDM Selection" Protocol, Security Model



XP PRINT SERVICE EXTENSION EVENTS

4.1 Issues

4.2 Overview

The Xp Print Protocol Extension to X introduces several new X events which can be selected using **`XpSelectInput`**, and can be received using the normal X event mechanisms (for example, **`XNextEvent`**).

4.3 Xp Print Event Summary

Table 4-1: Xp Print Extension Events

Event Mask for selection	Event Type(s) returned	Details in Event Struct	Event Structure
XPNoEventMask	**n.a.**	**n.a.**	**n.a.**
XPPrintMask	**XPPrintNotify**	**XPStartPageNotify**	**XPPrintEvent**
		XPStartDocNotify	
		XPStartJobNotify	
		XPEndPageNotify	
		XPEndDocNotify	
		XPEndJobNotify	
XPAttributeMask	**XPAttributeNotify**	**XPJobAttr**	**XPAttributeEvent**
		XPDocAttr	
		XPPageAttr	
		XPPrinterAttr	
		XPServerAttr	

The constants for the Xp Print event masks, types and details are defined in `<X11/Print.h>`. The structures for the Xp Print events are described below, and also defined in `<X11/Print.h>`.

4.4 Xp Print Event Details & Structures

The following describes the Xp Print Events in detail.

4.4.1 XPNoEventMask

When no events are wanted.

4.4.2 XPPrintMask

Asks the *Xp Print X-Server* to send notice when calls to **XpStartPage**, **XpStartDoc**, **XpStartJob**, **XpEndPage**, **XpEndDoc** and **XpEndJob** have actually been processed and completed.

```
typedef struct
{
      int          type;        /* base + XPPrintNotify */
      unsigned long serial;     /* # of last req processed by server */
      Bool         send_event; /* true if from a SendEvent request */
      Display      *display;    /* Display the event was read from */
      XPPrintContext context; /* print context where operation
                                  was requested */
      Bool         cancel;      /* was detailed event canceled */
      int          detail;      /* XPStartJobNotify, XPEndJobNotify,
                                  XPStartDocNotify, XPEndDocNotify,
                                  XPStartPageNotify, XPEndPageNotify */
} XPPrintEvent;
```


4.4.3 XPAttributeMask

Asks the *Xp Print X-Server* to send notice when any of the print attribute stores maintained by the *Xp Print X-Server* change. The *Xp Print X-Server* may have initiated the changes to the attributes, or a call to **XpSetAttributes** may have made the changes.

```
typedef struct
{
      int          type;        /* base + XPAttributeNotify */
      unsigned long serial;     /* # of last req processed by server */
      Bool         send_event; /* true if from a SendEvent request */
      Display      *display;    /* Display the event was read from */
      XPPrintContext context; /* print context where operation
                                  was requested */
      int          detail;      /* XPJobAttr, XPDocAttr, XPPageAttr,
                                  XPPrinterAttr, XPServerAttr */
} XPAttributeEvent;
```

4.5 Receiving Xp Print Events

As with all X extensions, returned extension event type values are computed relative to some base value. An example code fragment to decode a Xp Print Extension event is as follows:

```
XEvent  event;
int event_base, error_base;

/*
 * fetch an event
 */
XNextEvent( ptr_dpy, &event );

/*
 * fetch the offsets for the Xp Print Extension
 */
XpQueryExtension( ptr_dpy, &event_base, &error_base );

/*
 * decode
 */
switch( event.type - event_base ) {
      case XPPrintNotify: /* handler */
                        break;
      case XPAttributeNotify: /* handler */
                        break;
}
```

DT PRINT DIALOG MANAGER

5.1 Overview

PURPOSE

A Print Dialog Manager (PDM) is a process separate from the X Print Server and X Printing Application that provides printer-specific and spooler-specific setup GUIs. An application could choose to understand and display such GUIs itself, but is advised to offload the task to a PDM so that new printers and spoolers can be supported by providing new PDMs, rather than requiring changes to all applications.

Within the X Print Service, a standard "Print Dialog Manager Selection Protocol" is described in the functional specification for **`XpNotifyPdm`**. This protocol allows an X Printing Application to engage a PDM.

This chapter describes the CDEnext *implementation* of a PDM that uses the PDM Selection Protocol. The CDEnext implementation involves a Print Dialog Manager Daemon (PDMD) executable *dtpdmd* which is initially engaged by the protocol, and it then determines which PDM is needed and starts it on behalf of the application. Within CDEnext, the executable *dtpdm* is the general purpose PDM that the *dtpdmd* can start.

Printer Vendors can choose to introduce new GUIs by: 1) developing their own PDM implementation that conforms to the PDM Selection Protocol, or 2) develop a PDM that can be started by the *dtpdmd*, or 3) possibly (is vendor dependent) integrate new shared or dynamic libraries into the *dtpdmd*. We recommend approach #2.

DESCRIPTION

The *dtpdm* executable implements a reasonably general-purpose print dialog manager capable of providing dialogs suitable for a number of different printers, but specifically tuned to the needs of the two reference printers, the PCL based HP DeskJet 1600C, and the Postscript based Sun SPARCprinter 2. The *dtpdm* uses the attributes (see **`XpSetAttributes`**) of the particular printer to provide a limited amount of automatic configuration of the options displayed in its printer setup dialog. The *dtpdm's* job setup dialog is designed for use with the *lp* spooler.

The *dtpdmd* executable implements a PDM startup mechanism. This two-layer mechanism means that the PDM Selection Protocol, PDM selection and startup, and security concerns can be delt with by the *dtpdmd*, and that resulting PDMs called by the *dtpdmd* are left with the minimal and simple task of displaying GUIs.

5.2 Dt Print Dialog Manager Daemon - dtpdmd

5.2.1 Short description

A daemon process that uses the Print Dialog Manager Selection Protocol to start Print Dialog Managers.

5.2.2 Long description

NAME

dtpdmd - a daemon process that uses the PDM Selection Protocol to start PDMs.

SYNOPSIS

`dtpdmd [options]`

OPTIONS

`-d display` Specifies the display connection to an X-Server upon which an X-selection will be created and managed for requests. If specified, it overrides the environment variable XPDMDISPLAY.

`-a selection` Specifies an alternate X-selection name for the *dtpdmd* to create and manage. If specified, it overrides the environment variable XPDMSELECTION. By default, the selection name is `PDM_MANAGER`.

`-p pdm` Specifies an alternate PDM execution string to use if no other PDM execution string can be derived, usually from the Server Attribute `dt-pdm-command` from the X-Server. By default, the execution string is `"dtpdm"`. All execution strings are applied against the current search path.

`-P pdm` Specifies an alternate PDM execution string that overrides all other sources of such execution strings. All execution strings are applied against the current search path.

`-s` Specifies that the *dtpdmd* should turn on the security exchange portion of the PDM Selection Protocol. By default, the dtpdmd will not exchange security information with an application over the wire, so the appearance of “auto hosting” cannot be done.

`-l logfile` Specifies a file for the logging of errors and warnings. Entires are time-stamped and may be generated by the dtpdmd or any child PDM via stderr. The previous contents of the log file are destroyed. By default, `/dev/null` is used.

DESCRIPTION

The *dtpdmd* is a long-lived daemon process that receives *client* requests for a *PDM*, uses some lookup rules, and then starts an appropriate *PDM* to service the request. When the *PDM* finishes, control is returned to the *dtpdmd*, and the *dtpdmd* in turn responds to the *client* with final status.

The *dtpdmd* uses the following “PDMD/PDM Protocol” to communicate with the *PDM*. Communication “to” the *PDM* is done via a standardized command line and environment. Communication “from” the *PDM* is done via standardized exit codes.

PDMD/PDM Protocol

SYNOPSIS

```
dt-pdm-command [dt-pdm-options] -vdisplay vdpy -window vwid
                                -pdisplay pdpy -pcontext pcid
```

OPTIONS

`dt-pdm-command`	This pdm executable path is derived by the dtpdmd from either the -p or -P options.
`dt-pdm-options`	These options may have accompanied the dt-pdm-command, whether specified from the dtpdmd command line by the -p or -P options, or from other sources.
`-display vdpy`	Specifies the display connection to the Video X-Server.
`-window vwid`	Specifies the window id on the Video X-Server to which the pdm's dialogs should be posted as transient windows.
`-pdisplay pdpy`	Specifies the display connection to the Print X-Server.
`-pcontext pcid`	Specifies the print context id on the Print X-Server. Used by the pdm to gain access to the same print context as the requesting application.

ENVIRONMENT

Prior to starting a PDM, the dtpdmd may first modify the following environment variables:

XAUTHORITY	If the dtpdmd has come into possession of x-authority information that the PDM will need, the dtpdmd will set the XAUTHORITY environment variable so that the PDM will automatically have access to the proper x-authority information.

In addition, the dtpdmd may set a "locale hint" passed to it by the "PDM Selection Protocol" from the client prior to starting a PDM. On POSIX systems, setlocale(3C) will be used.

RETURN VALUES

The following integer constants are defined in <Dt/dtpdmd.h>:

`PDM_EXIT_OK`	The PDM is telling the PDMD that the user selected "OK" to dismiss the PDM.
`PDM_EXIT_CANCEL`	The PDM is telling the PDMD that the user selected "CANCEL" to dismiss the PDM.
`PDM_EXIT_VXAUTH`	The PDM is telling the PDMD that it did not have proper authority to make a display connection on the Video X-Server.
`PDM_EXIT_PXAUTH`	The PDM is telling the PDMD that it did not have proper authority to make a display connection to the Print X-Server.
`PDM_EXIT_ERROR`	The PDM is telling the PDMD that it encountered an error.

`all other values` All unknown return values, likely from uncontrolable exit conditions often found in other libraries (e.g. untrapped XIO errors from libX), will be equated by the PDMD to be the same as PDM_EXIT_ERROR.

The constant definitions in <Dt/dtpdmd.h> are as follows:

```
#define PDM_EXIT_OK     191   /* “OK” */
#define PDM_EXIT_CANCEL 192   /* “CANCEL” */
#define PDM_EXIT_VXAUTH 193   /* no print display authorization */
#define PDM_EXIT_PXAUTH 194   /* no video display authorization */
#define PDM_EXIT_ERROR  195   /* all other error reasons */
```

RETURN VALUES

ENVIRONMENT

FILES

SEE ALSO

5.3 Dt Print Dialog Manager

5.3.1 Short description

The dialog manager is a process separate from the print server. Its provides the printer-specific GUIs on behalf of a printing application

5.3.2 Long description

NAME

dtpdm - program invoked by *dtpdmd* to provide printer-specific GUIs.

SYNOPSIS

`dtpdm [options]`

OPTIONS

`-display vdpy`	Specifies the display connection to the Video X-Server.
`-window vwid`	Specifies the window id on the Video X-Server to which the pdm's dialogs should be posted as transient windows.
`-pdisplay vdpy`	Specifies the display connection to the Print X-Server.
`-pcontext pcid`	Specifies the print context id on the Print X-Server. Used by the pdm to gain access to the same print context as the requesting application.

DESCRIPTION

At an application's request *dtpdm* will post to the user's display a set of printer-specific dialogs enabling the user to configure a variety of printer options.

The *dtpdm* program provides a setup dialog to X printing applications that allows the user to set printer specific, and job specific options. Though the setup dialog will appear to be part of the application, it is actually managed by the *dtpdm* program on behalf of the application. It is capable of providing dialogs in all locales for which there exist applicable message catalogs.

dtpdm presents a dialog containing the printer name and description plus an XmNotebook widget. This notebook widget contains two tabs: one for the Printer Setup Box and one for the Job Setup Box. Each of these boxes provide controls that allow for configuration of various printing options. The *dtpdm* dialog also contains three pushbuttons labelled: "OK", "Cancel", and "Help". When the OK button is activated the dialog is dismissed and the newly configured printing options are set in the current print context (via **`XpSetAttributes`**). When the Cancel button is activated the dialog is dismissed and no changes are made to the print context.

The next two sections describe the setup boxes in greater detail.

CDEnext

Printer Setup Box

The Printer Setup box presents options specific to the currently selected printer. The options presented may vary in other PDM implementations. The following explains the printer setup options presented by *dtpdm*.

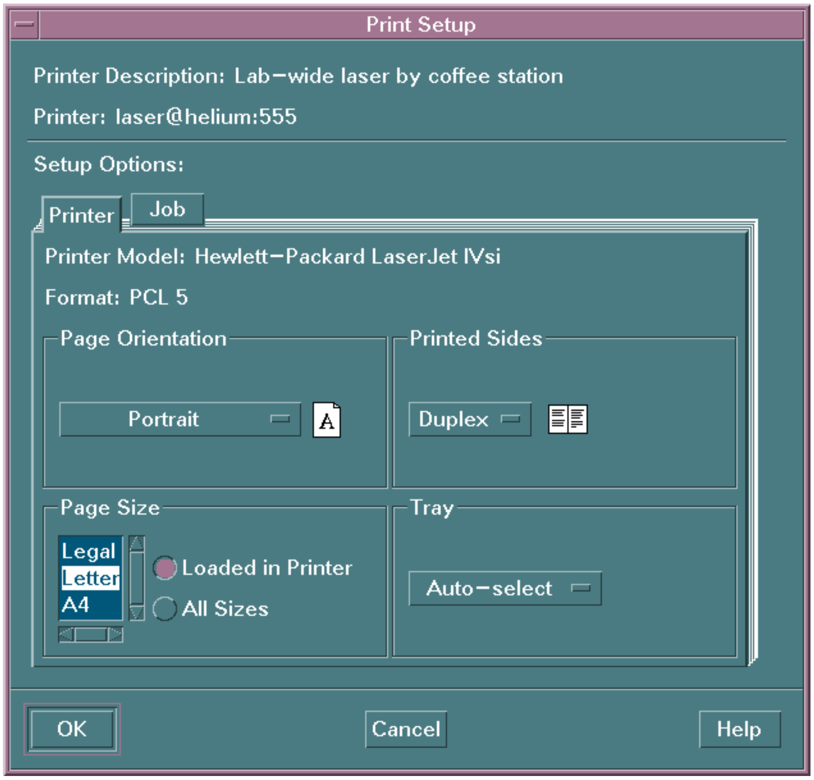


Figure 5-1. Printer Setup Box

Printer Information

This section of the setup box presents information about the X Printer. The information fields presented are the printer model and the document format used to generate documents sent to this X Printer.

Page Orientation

This options menu allows the user to select how the output will be oriented on the page. The orientation options presented in the menu depend on the printer, but up to four orientations are possible: portrait, landscape, reverse portrait and reverse landscape. An icon adjacent to the options menu is provided. This icon will present a graphical illustration appropriate for the current menu selection.

Printed Sides

This options menu allows the user to select single or double-sided printing. The actual choices available depend on the printer, but up to three choices are possible: simplex, duplex, and tumble. An icon adjacent to the options menu is provided. This icon will present a graphical illustration appropriate for the current menu selection.

Tray

This options menu allows the user to select which printer tray the media will be drawn from. The “Auto-select” tray option will be presented for all printers. Selecting this option indicates that the user has no preference as to which tray to use. Remaining entry possibilities are dependent on the printer.

Page Size

This list box allows the user to select the media size for printing. The entries presented in this list depend on the whether the “Loaded in Printer” or “All Sizes” radio button is selected.

Loaded in Printer

The user selects the “Loaded in Printer” radio button to view the media sizes currently available on the printer. If the current “Media Source” option is “Auto-select”, the user will see all media sizes available in all of the printer’s trays. If a specific tray is selected, only the media size loaded in that tray will be presented. Information on which media size is available in which tray is provided by the system administrator via the **`input-trays-medium`** attribute. If the system administrator does not provide this information, the “Loaded in Printer” radio button will be inactive.

All Sizes

The user selects the “All Sizes” radio button to view all supported media sizes available for the printer. When this button is selected, the Media Source options menu will contain only the “Auto-select” option. This button is provided for the following situations:

- in case the system administrator has not specified which sizes are loaded in the printer
- if a desired media size is not loaded, some printers can prompt for the requested size
- printing to a file, where the output may in fact never reach an actual printer

Job Setup Box

The Job Setup box presents options specific to the spooler controlling the printer. The options presented may vary depending on the spooler in other PDM implementations. The following explains the job setup options presented by *dtpdm*.

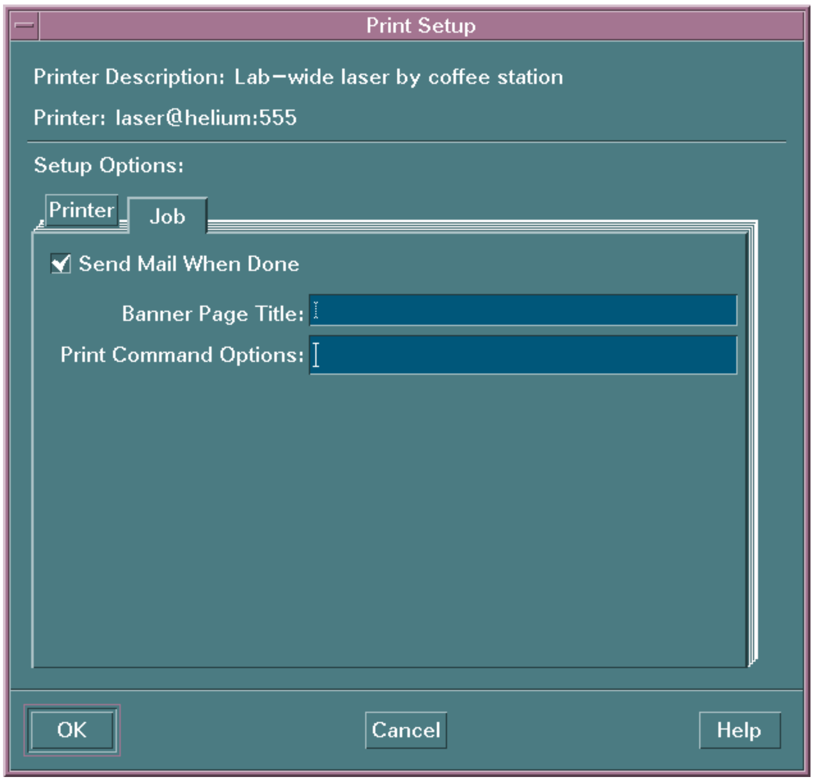


Figure 5-2. Job Setup Box

Send Mail When Done
When selected, an email message will be sent to the user from the spooler when the job is completed.

Banner
A text field into which the user can enter text which will appear on the banner page of the output.

Options
A text field into which the user can specify command line options and arguments that will be included in the command line used to invoke the spooler. No parsing of this field will be performed, and what are considered valid arguments is dependent entirely upon the underlying spooler.

CDEnext

STARTUP

The print dialog manager is started by the print dialog manager daemon, *dtpdmd*.

ENVIRONMENT

The Dt Print Dialog Manager uses the environment variable LANG to specify the location of its localized message file.

RETURN VALUES

The following integer constants are defined in <Dt/dtpdmd.h>:

`PDM_EXIT_OK`	The PDM is telling the PDMD that the user selected “OK” to dismiss the PDM.
`PDM_EXIT_CANCEL`	The PDM is telling the PDMD that the user selected “CANCEL” to dismiss the PDM.
`PDM_EXIT_VXAUTH`	The PDM is telling the PDMD that it did not have proper authority to make a display connection on the Video X-Server.
`PDM_EXIT_PXAUTH`	The PDM is telling the PDMD that it did not have proper authority to make a display connection to the Print X-Server.
`PDM_EXIT_ERROR`	The PDM is telling the PDMD that it encountered an error.
`all other values`	All unknown return values, likely from uncontrolable exit conditions often found in other libraries (e.g. untrapped XIO errors from libX), will be equated by the PDMD to be the same as PDM_EXIT_ERROR.

See the *dtpdmd* specification for additional information.

X PRINT CONFIGURATION DATABASES

6.1 Configuration Files Overview

Configuration files provide the raw information that is used by the X Print Service components. Strictly speaking, the configuration files, print dialog manager, and ddx drivers of the print server form a matched set. The configuration files, though, have been designed to be as flexible as possible.

PURPOSE

Provide configuration information for X Print Service components.

DESCRIPTION

Most of the configuration files are in the form of an XRM resource file. This provides maximum flexibility. The hierarchical nature of the data base avoids name clashes and wild cards can be used to signify that certain characteristics apply to many printers. Also, additional attributes can be added later.

This rest of this chapter will document the configuration directories and files used by the X Print Service.

$XPCONFIGDIR is an environment variable read by the X print server, that defines the root of the configuration directory hierarchy. If $XPCONFIGDIR is not defined, the server will default to `<XRoot>/lib/X11`, where `<XRoot>` is the root of the X11 install tree. Determining configuration values is performed as follows:

1. Search $XPCONFIGDIR`/C/print` to obtain default values from a configuration file.
2. If the configuration file is not found, server-defined defaults will be used.
3. For locales other than C, search $XPCONFIGDIR/$LANG`/print` and use the configuration file values to augment the defaults determined above.

One exception to this is the `Xprinters` file. This file indicates which printers will be managed by the X Print Server. The path and name of this file is indicated by the `-XpFile` command line option defined by the X Print Server. If the command line option is not present, the X Server will default to $XPCONFIGDIR`/C/print/Xprinters`. This file is optional.

There are several types of configuration files. Within several subdirectories:

- A file that indicates which printers will be managed by the X Print Server. This file is referred to as the `Xprinters` file.
- Printer attributes files that define the capabilities of the printer model. The name of the file is typically all uppercase, and consists of the manufacturer and the model of printer. Examples of file names are: `HPDJ1600C`, `IBM-4039-16l`, and `SUN-NP20`.

CDEnext

- Printer attribute files that define the capabilities of printers installed on a particular X Print Server.
- Job and document attribute files that specify initial values for the print operation.
- Optional ddx driver configuration files. The format of each file is internal to the corresponding ddx driver.

The encoding for most of the configuration files documented in this chapter is Compound Text as defined by the X Window System. The only exception is the optional ddx driver configuration files. These files are defined at the discretion of the driver developer.

6.2 Configuration Directories

6.2.1 Print Configuration Directory

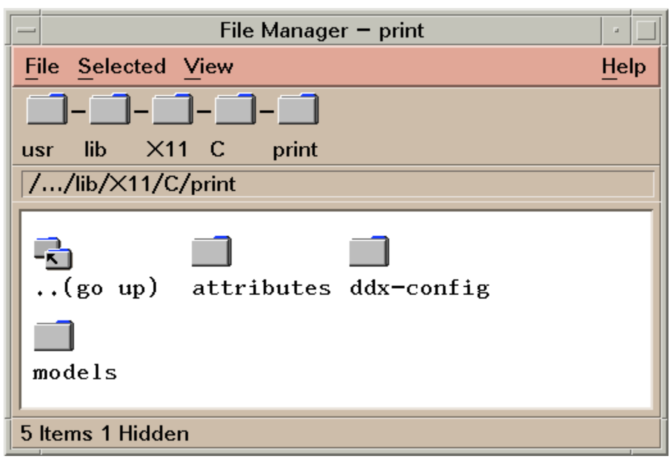


Figure 6-1. Example Print Configuration Directory

The X Print Service configuration directory is assumed to be `/usr/lib/X11/C/print` for the purposes of this discussion. The configuration files will in actual use be distributed throughout the configuration hierarchy, as described in the “Configuration Files Overview” section.

At the top level of the locale specific `print` directory, three subdirectories are defined. The `ddx-config` directory contains configuration information specific to X Print Server ddx drivers. The `models` directory defines default attributes and internal font metrics for various models of printers. The `attributes` directory defines attributes for the various X Printers defined on the host system. The following sections describe these directories in more detail.

6.2.2 Printer Model Configuration Directories

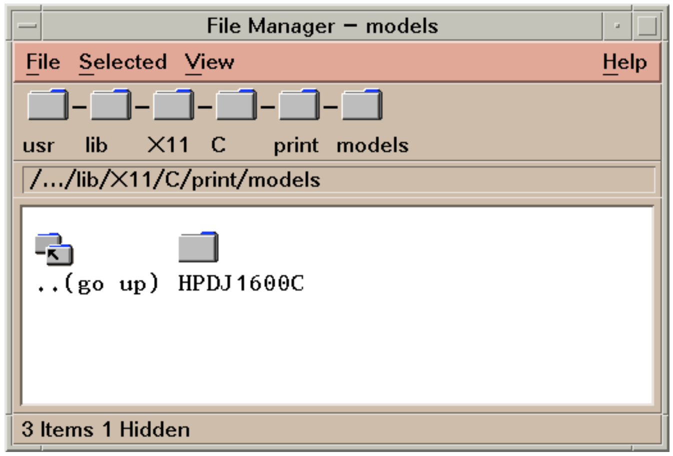


Figure 6-1. Example X Printer `models` Directory

The `models` directory contains subdirectories that define configuration information for various models of printers. Each subdirectory corresponds to a specific printer model or a specific class of printer models. The names of these model directories define valid values for the **`xp-model-identifier`** attribute in the printer attributes file. See the "Printer Attribute Definitions" section in the "X Print Service Attributes" chapter for more information on this attribute.

It is recommended that only uppercase characters be used for the names of model configuration directories. This will help avoid namespace collisions between model names and printer names, when these names are used as qualifiers in the attributes files. See the "Printer Attributes File", "Document Attributes File", and the "Job Attributes File" sections in this chapter for information on the format of these files.

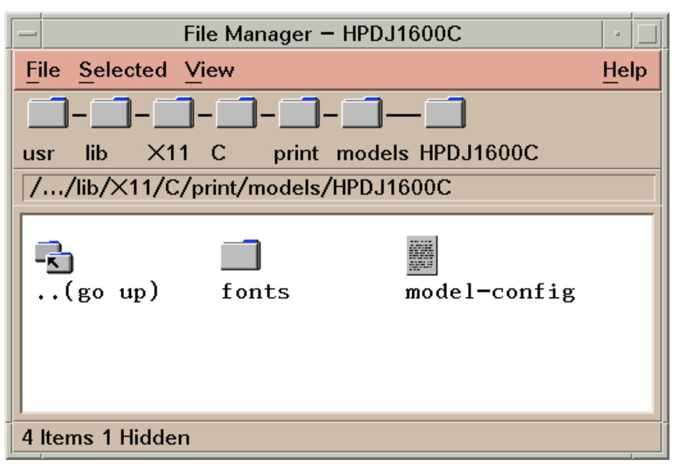


Figure 6-2. Example Printer Model Configuration Directory

The printer model configuration directory contains a `model-config` file and a `fonts` directory. The `model-config` file defines a set of default attributes for a specific printer model or a specific class of printer models. See the “Printer Model Attributes File” section for details on the format of this file.

The `fonts` directory defines font metrics for the printer’s internal fonts. If any fonts are defined under a locale-specific subdirectory, they obscure all fonts defined under the default C locale subdirectory.

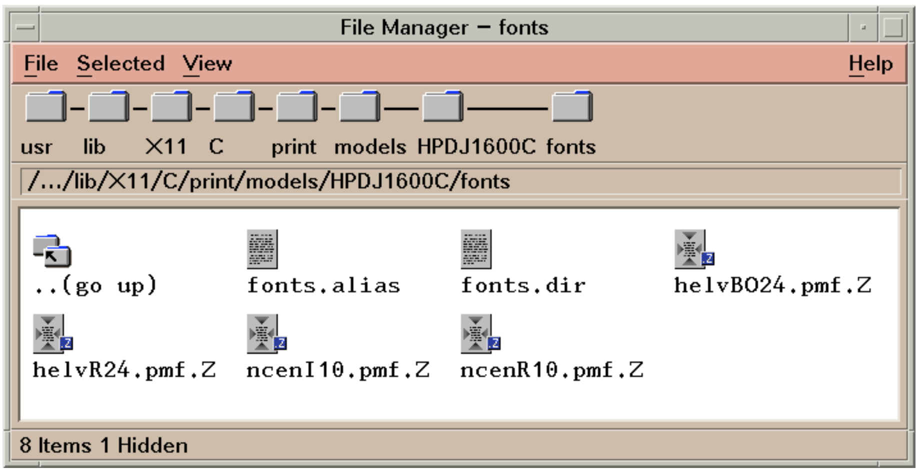


Figure 6-3. Example X Printer Internal Fonts Directory

The `fonts` directory is read by the X Print Server. See the “Fonts” chapter for more information.

6.2.3 Printing Attributes Configuration Directory

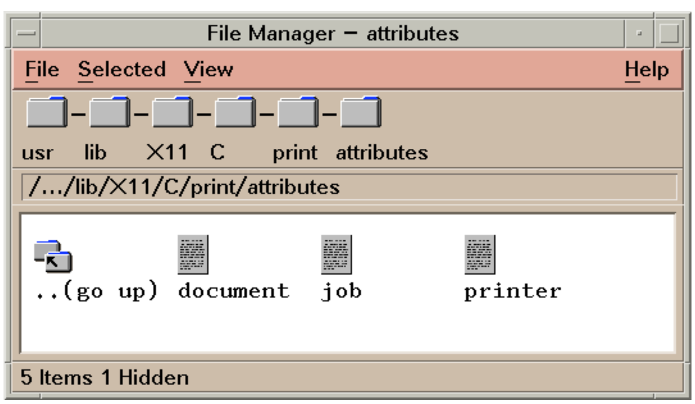


Figure 6-1. Printing Attributes Configuration Directory

The files in the `attributes` directory contain initial values for the X Print Service attributes. These attributes define print setup options (`document` and `job`) and provide printer capabilities (`printer`). See the “Printer Attributes File”, “Document Attributes File”, and the “Job Attributes File” sections in this chapter for information on the format of these files.

6.2.4 ddx Driver Configuration Directories

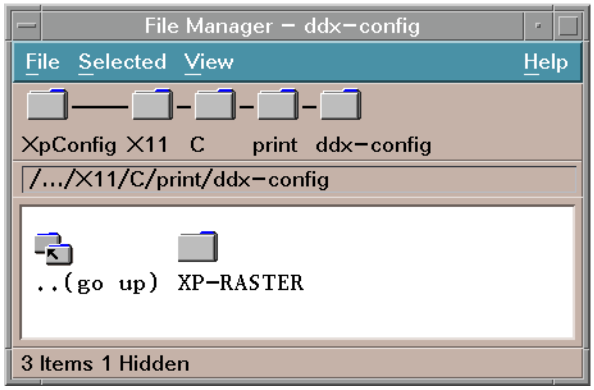


Figure 6-1. Example `ddx-config` Directory

The `ddx-config` directory contains ddx driver configuration directories. A ddx driver may or may not require one of these directories. The contents of each directory is specific to the corresponding driver. The name of the directory is the same as the driver name provided by the ddx driver to the X Print Server, and is also used as the value of the **`xp-ddx-identifier`** printer attribute.

Figure 6-2. shows an example of the ddx driver configuration directory for the raster driver.

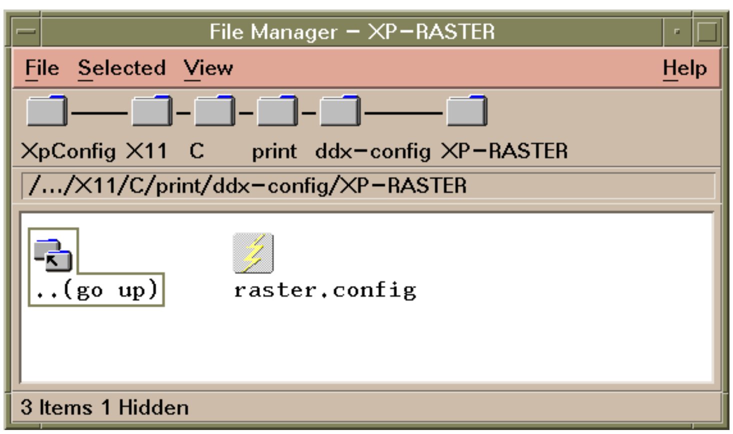


Figure 6-2. Example ddx Driver Configuration Directory

Driver configuration files in this directory may be assigned on a per-printer basis by using the **`xp-ddx-config-file-name`** printer attribute. Whether or not this attribute is utilized is determined by each individual driver.

CDEnext

6.3 Xprinters File

NAME

Xprinters file - identify printers to be managed by an X Print Server

DESCRIPTION

The `Xprinters` file is read by an X Print Server during initialization in order to determine which printers it will manage.

Lines in the file consist of a keyword followed by a value. Keyword recognition is case-sensitive. Any data following the comment character “#” on a given line is ignored.

The encoding for the `Xprinters` file is Compound Text as defined by the X Window System.

KEYWORDS

`Augment_Printer_List`

This keyword is used to generate a list of printer names that will be added to the list of printers the server will manage. If this line is not specified, or if the Xprinters file does not exist, the server will generate a list of printers by utilizing the output of *lpstat(1)*.

Predefined values for the `Augment_Printer_List` keyword are:

`%default%` Explictly invoke the default behavior, i.e. augment the list of printers by utilizing the output of *lpstat(1)*.

`%none%` Do not augment the list of printers. This provides a way to override the default behavior of calling *lpstat(1)* when no `Augment_Printer_List` line is present.

In addition, the value may be specified as a POSIX shell command pipeline that generates a list of printers on **`stdout`**. This generated list is added to the list of printers managed by the server.

`Printer`

A whitespace delimited list of one or more printer names to add to the list of printers managed by the server.

`Map`

The attributes configuration files utilize a printer qualifier, defined by the X Print Server, that is the printer name by default, provided the characters comprising the printer name conform to the restricted set of characters allowed for the printer qualifier, that is, the set of characters allowed for Xrm resource names. The `Map` keyword is provided to allow specification of a printer qualifier when a default printer qualifier is not generated by the server, or if an override of the default qualifier is desired.

The `Map` value is of the form *<printer name> <printer qualifier>*, for example:

```
Map könig koenig
```

EXAMPLE

```
##########################################################################
#
# Xprinters sample configuration file
#
# The Xprinters file is read by an X Print Server during initialization in
# order to determine which printers it will manage. The actual file name and
# path is given to the X Print Server via the -XpFile command
# line option.
##########################################################################

##########################################################################
# Use lpstat to augment the list of printers managed by the
# server. (This is the default behavior if the Xprinters file is
# not specified, or if an "Augment_Printer_List" line is not specified.)
##########################################################################
Augment_Printer_List %default%

##########################################################################
# Use the specified command pipeline to augment the list of printers
# managed by the server.
##########################################################################
#Augment_Printer_List lpstat -a | cut -d " " -f 1  #equivalent to default

##########################################################################
# Do not augment the list of printers managed by the server.
##########################################################################
#Augment_Printer_List %none%

##########################################################################
# Add individual printers to the list of printers managed by the
# server.
##########################################################################
#Printer laser_1 laser_2 laser_c4
#Printer deskJet_1 deskJet_2
#Printer xpress

##########################################################################
# Provide printer qualifiers for non-conforming printer names
##########################################################################
Map könig koenig
```

SEE ALSO

- *lpstat(1)*

6.4 Printer Model Attributes File

NAME

printer model attributes file - printer model capabilities

DESCRIPTION

The printer model attributes file consists of printer attributes for a specific printer model or a specific class of printer models. This file is delivered by a printer vendor or ddx printer driver developer in order to provide default configuration information for a printer.

Valid attributes are based on a subset of the POSIX 1387.4 Printer Object attribute definitions (*note:* the X Print Service is *not* an implementation of POSIX 1387.4). See the "Printer Attribute Definitions" section in the "X Print Service Attributes" chapter for the complete list.

The encoding for the printer model attributes file is Compound Text as defined by the X Window System.

Attribute names must be qualified using either the **`xp-model-identifier`** or an asterisk (`*`). For example, if `HPDJ1600C` is the **`xp-model-identifier`**, then to initialize the **`plexes-supported`** attribute to **`simplex`**, use: `HPDJ1600C.plexes-supported: simplex`. For the asterisk, use: `*.plexes-supported: simplex`. If the same attribute is specified using each method, the **`xp-model-identifier`** qualified entry takes precedence.

EXAMPLE

```
! This is the configuration file for the HP DeskJet 1600C printer.
! It is designed for use with the CDEnext Sample Implementation
! PCL, raster drivers, and print dialog manager.

HPDJ1600C.printer-model: Hewlett-Packard DeskJet 1600C
HPDJ1600C.descriptor: Hewlett-Packard DeskJet 1600C
HPDJ1600C.printer-resolutions-supported: 300
HPDJ1600C.content-orientations-supported: portrait landscape
HPDJ1600C.document-formats-supported: {PCL 5}
HPDJ1600C.plexes-supported: simplex
HPDJ1600C.xp-ddx-identifier: XP-PCL
HPDJ1600C.xp-embedded-formats-supported: {PCL 5} {HPGL 2}
HPDJ1600C.dt-pdm-command: dtpdm

! na-letter, iso-a4, na-legal, na-number-10-envelope, more?
! assumes 1/4" unprintable margins for all media
HPDJ1600C.medium-source-sizes-supported: \
{'' \
  {na-letter FALSE {6.35 209.55 6.35 273.05}} \
  {iso-a4 FALSE {6.35 203.65 6.35 290.65}} \
  {na-legal FALSE {6.35 209.55 6.35 349.25}} \
  {na-number-10-envelope FALSE {6.35 222.25 6.35 98.425}} \
}
```

CDEnext

SEE ALSO

- The “Printer Attributes File” section in this chapter.
- The “Printer Attribute Definitions” section in the “X Print Service Attributes” chapter.

6.5 Printer Attributes File

NAME

printer attributes file - printer configuration

DESCRIPTION

The printer attributes file identifies capabilities and defaults for an X printer on the host system. This file is defined by the system administrator. Definitions in this file override attributes defined in the Printer Model Attributes file.

Valid attributes are based on a subset of the POSIX 1387.4 Printer Object attribute definitions. See the “Printer Attribute Definitions” section in the “X Print Service Attributes” chapter for the complete list.

The encoding for the printer attributes file is Compound Text as defined by the X Window System.

Attribute names must be qualified by using one of the following (listed in order of precedence):

printer qualifier
: Set this attribute for the printer indicated by the printer qualifier. The set of valid printer qualifiers is defined as the list of printer qualifiers managed by the X Print Server (the server typically generates this list by reading the `Xprinters` file).

  Example: `dj_1.document-formats-ready: {PCL 5}`

xp-model-identifier
: Set this attribute for all printers of a specific model.

  Example: `HPDJ1600C.document-formats-ready: {PCL 5}`

*****
: Set this attribute for all printers.

  Example: `*.document-formats-ready: {PCL 5}`

EXAMPLE

```
*.xp-model-identifier: HPLJ4SI

HPDJ1600C.input-trays-medium: { main na-letter }

deskJet_1.descriptor: DeskJet 1600C in Bob’s Cubicle
deskJet_1.xp-model-identifier: HPDJ1600C

laser_1.descriptor: 4si in Brock’s Bay
laser_1.input-trays-medium: {top na-letter} {bottom na-legal} \
    {large-capacity na-letter}
laser_2.descriptor: laserjet in test area
laser_2.plexes-supported: simplex
laser_2.input-trays-medium: {top iso-a4} {bottom iso-a4}
```

SEE ALSO

- The “Printer Model Attributes File” section in this chapter.
- The “Printer Attribute Definitions” section in the “X Print Service Attributes” chapter.

6.6 Job Attributes File

NAME

job attributes file - print job initial values

DESCRIPTION

The encoding for the job attributes file is Compound Text as defined by the X Window System.

Attribute names must be qualified by using one of the following (listed in order of precedence):

printer qualifier
Set this attribute for the printer indicated by the printer qualifier. The set of valid printer qualifiers is defined as the list of printer qualifiers managed by the X Print Server (the server typically generates this list by reading the `Xprinters` file).

Example: `laser_1.job-name: Payroll Reports`

xp-model-identifier
Set this attribute for all printers of a specific model.

Example: `HPDJ1600C.job-name: Payroll Reports`

*****
Set this attribute for all printers.

Example: `*.job-name: Payroll Reports`

EXAMPLE

```
! defaults
*.job-name:
*.notification-profile: {}

! Printer laser_1 prints paychecks - always send email on completion
laser_1.notification-profile: {{event-report-job-completed} electronic-mail}
laser_1.job-name: Payroll Reports
```

6.7 Document Attributes File

NAME

document attributes file - print document initial values

DESCRIPTION

The encoding for the document attributes file is Compound Text as defined by the X Window System.

Attribute names must be qualified by using one of the following (listed in order of precedence):

printer qualifier
Set this attribute for the printer indicated by the printer qualifier. The set of valid printer qualifiers is defined as the list of printer qualifiers managed by the X Print Server (the server typically generates this list by reading the `Xprinters` file).

Example: `dj_1.plex: duplex`

xp-model-identifier
Set this attribute for all printers of a specific model.

Example: `HPDJ1600C.plex: duplex`

*****
Set this attribute for all printers.

Example: `*.plex: duplex`

EXAMPLE

```
*.default-input-tray: top
*.default-printer-resoution: 300
*.plex: duplex
*.content-orientation: portrait
*.copy-count: 1
*.document-format: {PCL 5}
HPLJ4SI.default-printer-resolution: 600
printer_1.default-input-tray: large-capacity
deskJet_1.plex: simplex
```

6.8 ddx Driver Configuration Files

NAME

ddx configuration file - ddx driver defined configuration

DESCRIPTION

The ddx configuration file is defined at the discretion of the ddx driver developer. The format of the information defined in the file is internal to the ddx driver. The developer may choose to publish the format of this file to allow for customization by system administrators.

EXAMPLES

The Raster driver supplied with the X Print Service utilizes a ddx configuration file. Here is an example of how it is defined:

```
! Raster ddx print driver configuration file
*PageCommand: command -o option
```

SEE ALSO

- The “X Print Driver Interface” chapter.

X PRINT SERVICE ATTRIBUTES

7.1 Overview

Printing-specific attributes play a key role in the X Print Service. They provide a general-purpose mechanism for storing information associated with printing. This information includes user print setup options and printer capabilities.

PURPOSE

Convey capabilities of print servers and printers, and options for print jobs and documents.

DESCRIPTION

The X Print Service selects attributes in a way that is consistent with X Windows, ISO/IEC 10175 (ISO DPA), and POSIX 1387.4 print standards (*note:* the X Print Service is *not* an implemention of the ISO DPA or POSIX 1387.4).

Applications typically will not have to deal with these attributes. The primary use of the attributes is for communication between the Print Dialog Manager and the printer ddx drivers. Generally, only applications that present their own job and printer setup dialogs will need to examine attribute information.

The initial values of X Print Service attributes are specified in the X Print Configuration Files. The configuration files are read by the X Print Server. The attributes are interpreted by each driver, and for each printer a set of relevant attributes is maintained. The Print Dialog Manager obtains the attribute set for a given print context and uses the values to initialize the job and printer setup dialogs. The user interacts with the dialogs, and sets values for the printing task. The Print Dialog Manager informs the server of the updated attribute values. When the printing task commences, the printer ddx driver reads the set of values to determine how it will render the print documents and submit the job.

The ISO DPA defines a number of abstract objects that are managed and manipulated during the printing process. These are known as DPA-Objects. Each DPA-Object is represented by a set of attributes which characterize that object. Each attribute in turn is composed of an attribute-type (attribute name) and zero or more attribute-values.

The X Print Service utilizes selected DPA-Objects, and for each of these, a subset of the associated attributes. The DPA-Objects used are:

Server Object Specifies attributes defined for the print server.

Job Object Specifies attributes for a single print request as sent to the spooler.

Document Object Specifies attributes used to define a single document within a job. If supported by the implementation, one or more documents may be submitted within a given job.

Printer Object Specifies attributes that identify printer capabilities.

The X Print Service also provides for changing certain document attributes on a page-by-page basis. This is a capability for which the ISO DPA does not define a separate DPA-Object. This set of attributes is known within the X Print Service as Page Attributes.

The X Print Service requires some additional attributes that are not defined by the ISO DPA. The attribute names for these attributes are prefixed with "**`xp-`**". The CDEnext implementation also requires some additional attributes that are not defined by the ISO DPA. The attribute names for these attributes are prefixed with "**`dt-`**"

This chapter defines the following sets of attributes for the X Print Service:

- Server Attributes
- Printer Attributes
- Job Attributes
- Document Attributes
- Page Attributes

7.2 Attribute Value Defaulting And Validation

This section provides an overview of default attribute values and validation of attribute values within the X Print Service. Details for individual attributes can be found in the rest of this chapter.

7.2.1 Defaulting Attribute Values

An attribute specification with an empty value shall indicate that the attribute has no value. Within X Print Service configuration files and attribute pools, an attribute specification that omits the value is effectively treated as if there is no attribute specification. An empty valued attribute specification that has precedence over a non-empty attribute specification (for instance, an empty printer qualified attribute over a non-empty model qualified attribute) will effectively "unset" the lower precedence attribute specification. When a print job commences, the X Print Service may infer a default value for an attribute that has no value. In some cases the X Print Service may explicitly assign a default value to an attribute before presenting it in an attribute pool.

The ISO DPA provides for explicitly assigning **`generic-none`** to an attribute in order to prevent any action implied for that attribute; however this implementation of the X Print Service does not recognize **`generic-none`** for any of the attributes it defines.

7.2.2 Validation of Attribute Values

The X Print Server (in conjunction with the print ddx drivers) ensures that attribute pools presented to the client are always comprised of valid attribute specifications, for attributes defined by the X Print Service. Validation is first performed when a print context is created for a particular printer. Validation is also performed whenever a client requests an update to an attribute pool.

Validation involves checking the attribute value against the set of valid values specified for the attribute. Validation may also take into account the current values of other attributes and the capabilities of the ddx driver.

At print context creation time, if the server determines that an attribute value is invalid, its course of action is determined based upon whether the attribute has a single value or a multiple value. For single valued attributes the server will reject the invalid attribute specification, and may decide to set an explicit default for the attribute in the pool. For multi-valued attributes, the server will reject each value component that is invalid. If all of the specified components are invalid, the server will reject the attribute specification, and for certain attributes will set an explicit default for the attribute in the pool.

When the client requests an update to an attribute pool (e.g. when calling **`XpSetAttributes`**), if the server determines that an attribute value is invalid, its course of action is the same as at print context creation time, with one exception; in the cases where the server would choose to use a default, the server will instead retain the pre-existing attribute specification found in the pool.

The server will provide log messages when invalid attributes are encountered when constructing an attribute pool. Print clients will not receive notification of invalid attribute specifications. Interested clients must re-read the attribute pool to determine if the requested value was accepted by the server.

It is important to note that as part of the validation for a given attribute a ddx driver may choose to alter other attributes in response to the change. For example one can imagine that changing the value of the **`document-format`** attribute would cause the value of the **`xp-embedded-formats-supported`** attribute to change as well. In spite of the fact that the sample implementation does not do this for any attribute, applications should be prepared for the value of an attribute to change in response to the changing of some other attribute's value.

7.3 Server Attribute Definitions

7.3.1 Description

Server attributes describe the X Print Server. These attributes are created by the X Print Server, and are not defined in any configuration files, nor are they redefined by the printer ddx drivers. Applications retrieve these attributes using **XpGetAttributes** from the X Print Extension API.

The following table shows where querying a server attribute value is supported within the X Print Service.

Table 7-1: Server Attribute Usage

Attribute	Configuration	DDX Driver	Application
document-attributes-supported		X	
job-attributes-supported		X	
locale		X	X
multiple-documents-supported		X	X

7.3.2 Server Attributes

document-attributes-supported

A list of document attributes supported by the X Print Server. This list is comprised of a set of whitespace-delimited attribute names.

The list of document attributes shall include only attributes that are handled by the X Print Server. The full set of supported document attributes for a given printer is determined by the printer ddx driver. The driver augments the value of this server attribute, and presents the full set of supported document attributes as the value of the Printer object **document-attributes-supported** attribute. As such, applications can only query the Printer attribute and not this Server attribute in order to determine which document attributes can be used.

job-attributes-supported

A list of the job attributes supported by the X Print Server. This list is comprised of a set of whitespace-delimited attribute names.

The list of job attributes shall include only attributes that are handled by the X Print Server. The full set of supported job attributes for a given printer is determined by the printer ddx driver. The driver augments the value of this server attribute, and presents the full set of supported job attributes as the value of the Printer object **job-attributes-supported** attribute. As such, applications can only query the Printer attribute and not this Server attribute in order to determine which job attributes can be used.

locale

The value of this attribute is the locale in which the X Print Server is running.

`multiple-documents-supported`

This attribute indicates whether the server supports jobs containing multiple documents. The sample implementation does not support multiple documents, so this value will always be **`False`** in the sample implementaion.

7.4 Printer Attribute Definitions

7.4.1 Description

Printer attributes describe printer capabilities. Applications retrieve these attributes using **XpGetAttributes** from the X Print Extension API.

The following table shows where querying and / or setting a printer attribute value is supported within the X Print Service (*note:* applications cannot set printer attribute values).

Table 7-1: Printer Attribute Usage

Attribute	Configuration	DDX Driver	Application
content-orientations-supported	X	X	X
descriptor	X	X	X
document-attributes-supported		X	X
document-formats-supported	X	X	X
dt-pdm-command	X	X	X
input-trays-medium	X	X	X
job-attributes-supported		X	X
medium-source-sizes-supported	X	X	X
plexes-supported	X	X	X
printer-model	X	X	X
printer-name		X	X
printer-resolutions-supported	X	X	X
xp-ddx-config-file-name	X	X	
xp-ddx-identifier	X	X	
xp-embedded-formats-supported	X	X	X
xp-listfonts-modes-supported	X	X	X
xp-model-identifier	X		
xp-page-attributes-supported		X	X
xp-raw-formats-supported	X	X	X
xp-setup-proviso	X	X	X
xp-spooler-command	X	X	X

CDEnext

7.4.2 Printer Attributes

content-orientations-supported

A list of orientations that the printer supports. The list is a group of strings separated by white space. Valid values are **portrait**, **landscape**, **reverse-portrait**, and **reverse-landscape**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above.

The initial value of the **content-orientations-supported** attribute is typically set by the printer vendor in the `model-config` file.

descriptor

The **descriptor** is a human readable description of the printer encoded as COMPOUND_TEXT. This description may contain more than one line. Some GUI components may choose to only display the first line due to space limitations.

No default is provided for this attribute. No validation of the attribute value is performed.

The initial value of the **descriptor** attribute is typically set by the system administrator in the `printer` attributes file.

document-attributes-supported

A list of document attributes supported for the printer. This list is comprised of a set of whitespace-delimited attribute names.

The value of the **document-attributes-supported** attribute is determined by the print ddx driver.

document-formats-supported

A list of document formats, including format variants and format versions that the print ddx driver supports. Each entry in the list is a structure comprised of the document-format, an optional document-format-variant, and an optional document-format-version. Specific printer ddx drivers may require specification of the optional values. Structure values are enclosed by curly braces "{}" and delimited by whitespace. Valid values in the sample implementation are **{ PCL 5 }** and **{ PostScript 2 }**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above. The actual set of valid document-format values varies based on the ddx.

The initial value of the **document-formats-supported** attribute is typically set by the printer vendor in the `model-config` file.

dt-pdm-command

The command that will be used to run the Print Dialog Manager for this printer. Command line options may be included, and will be used (unexpanded) by the DtPDM daemon when invoking the PDM.

Example: **dtpdm -option n**

CDEnext

The default value is implicitly determined by the DtPDM daemon to be **dtpdm**. No validation of the attribute value is performed.

The initial value of the **dt-pdm-command** attribute is typically set by the printer vendor in the `model-config` file.

input-trays-medium

This attribute identifies what medium is loaded in each printer tray. The value is specified as a list of structures, each of which contains a tray identifier and a medium identifier. Valid tray identifiers are **top**, **middle**, **bottom**, **envelope**, **manual**, **large-capacity**, **main**, and **side**. The X Print Service defines valid medium identifiers to be the standard values of the **medium-size** attribute as specified in ISO/IEC 10175-1.

Example: **{top na-letter} {bottom iso-a4}**

The default value is implicitly determined to be an empty list. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above. Additionally, for each tray / medium (size) combination, the tray must be present in the value of the **medium-source-sizes-supported** attribute, and the medium size must be listed for that tray; otherwise the tray / medium combination is considered invalid.

The initial value of the **input-trays-medium** attribute is typically specified by the system administrator in the `printer` attributes file.

job-attributes-supported

A list of the job attributes supported for the printer. This list is comprised of a set of whitespace-delimited attribute names.

The value of the **job-attributes-supported** attribute is determined by the print ddx driver.

medium-source-sizes-supported

This attribute identifies or specifies the sizes of media that are supported by the printer. For each input tray a set of supported media sizes is indicated. For each medium, the page size, an indicator as to the medium feed direction, and the assured reproduction area the printer supports are specified.

Valid input tray values are **top**, **middle**, **bottom**, **envelope**, **manual**, **large-capacity**, **main**, and **side**. If the printer has only one input tray, specification of this value is optional (a placeholder of **''** is required).

The page size is a descriptive-name indicating the size of the page. Examples are **iso-a4**, **na-letter**, and **na-legal**. The complete list of valid values is the set of descriptive-names defined for the standard values of the **medium-size** attribute as specified in ISO/IEC 10175-1.

The medium feed direction is represented as a boolean value indicating whether the long edge (**TRUE**) or the short edge (**FALSE**) feeds into the printer so that orientation is specified.

The assured reproduction area is the area within the current medium that the printer can render to. This area is specified in millimeters according to the RCS coordinate system defined by the ISO DPA. The X Print Service requires that each position be specified as an integer. The area value is defined by a structure containing the minimum-x, maximum-x, minimum-y, and maximum-y. For example, if the printer cannot print within one centimeter of the edges of A4 paper, then the value would be:
`{ 10 200 10 287 }`.

The value for a medium size is specified in a structure comprised of the page size, the feed direction indicator, and the assured reproduction area. For the A4 example, if the short edge of the medium feeds into the printer the value would be:
`{ iso-a4 FALSE {10 200 10 287} }.`

The value of the **`medium-source-sizes-supported`** attribute is a list of structures, each comprised of the input tray value and a set of medium size values. For example, if the printer has two input trays which each support A4 or A5 paper, and the short edge of the medium feeds into the printer, and the printer cannot render within one centimeter of the medium edges, then the value of the **`medium-source-sizes-supported`** attribute would be:
`{ top {iso-a4 FALSE {10 200 10 287}} {iso-a5 FALSE {10 138 10 200}} } { bottom {iso-a4 FALSE {10 200 10 287}} {iso-a5 FALSE {10 138 10 200}} }`.

The default value is explicitly set with an omitted input tray, a single medium size of **`na-letter`**, short edge feed direction, and a reproducible area based on 1/4 inch margins. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above. Syntax errors may cause the entire value to be considered invalid.

The initial value of the **`medium-source-sizes-supported`** attribute is typically set by the printer vendor in the `model-config` file.

`plexes-supported`

A list of plex options that the printer supports. The list is a group of strings separated by white space. Valid values are **`simplex`**, **`duplex`**, and **`tumble`**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above.

The initial value of the **`plexes-supported`** attribute is typically set by the printer vendor in the `model-config` file.

`printer-model`

Human-readable text that identifies the make and model of the printer. This value is encoded as COMPOUND_TEXT.

Example: **`Hewlett-Packard LaserJet IV`**

No default is provided for this attribute. No validation of the attribute value is performed.

The initial value of the **`printer-model`** attribute is typically set by the printer vendor in the `model-config` file.

printer-name

This attribute uniquely identifies a printer on a given X Print Server. This attribute is not explicitly set in a configuration file; it is generated by the X Print Server.

printer-resolutions-supported

A list of the resolutions in dots per inch that the printer supports. For example, if a printer supports 300 dpi and 600 dpi printing, the value would be: **`300 600`**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above.

The initial value of the **`printer-resolutions-supported`** attribute is typically set by the printer vendor in the `model-config` file.

xp-ddx-config-file-name

The name of a ddx driver-defined configuration file. Whether or not this attribute is utilized is determined by each individual driver. The file name is taken relative to the ddx configuration directory for the driver.

A default value may be assumed depending on the individual driver.

The initial value of the **`xp-ddx-config-file-name`** attribute is typically set by the printer vendor in the `model-config` file.

xp-ddx-identifier

This attribute identifies which printer ddx driver should be used for this printer. The value is a driver name provided by the ddx driver to the server, and is determined by the printer driver developer. It is recommended that the value consist of the manufacturer and either the PDL used, for generic drivers, or a printer model, for model-specific drivers. Valid values in the sample implementation are **`XP-PCL`**, **`XP-POSTSCRIPT`**, and **`XP-RASTER`**.

The default value in the sample implemention is implictly taken to be **`XP-POSTSCRIPT`** by the X Print Server. Validation for this attribute is as described for single valued attributes in 7.2.2 above.

The initial value of the **`xp-ddx-identifier`** attribute is typically set by the printer vendor in the `model-config` file.

xp-embedded-formats-supported

This attribute identifies the set of data formats recognized as valid values for the *`doc_fmt`* parameter of the **`XpPutDocumentData`** function, when this function is called within a print document of *`type`* **`XPDocNormal`**. See the “XpStartDoc - XpEndDoc - XpCancelDoc” and “XpPutDocumentData” sections of the “X Print Service Extension Library” chapter for details.

The value is a list of data formats. Each entry in the list is a structure comprised of the data format, an optional format variant, and an optional format version. Specific printer ddx drivers may require specification of the optional values. Structure values are enclosed by curly braces “{}” and delimited by whitespace. Valid values are defined by the printer ddx driver. For the sample implementation, valid values may include **`{EPS}`**, **`{PostScript 2}`**, **`{PCL 5}`**, and **`{HPGL 2}`**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above. The actual set of valid document-format values varies based on the ddx.

The initial value of the **xp-embedded-formats-supported** attribute is typically set by the printer vendor in the `model-config` file.

xp-listfonts-modes-supported

Defines the set of values that may be used to comprise the value of the **xp-listfonts-modes** document / page attribute. The value is a whitespace delimited list of listfonts mode values, which are defined below. See the documentation for the **xp-listfonts-modes** attribute for details.

Valid listfonts mode values in the sample implementation are **xp-list-internal-printer-fonts** and **xp-list-glyph-fonts**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above.

The initial value of the **xp-listfonts-modes-supported** attribute is typically set by the printer vendor in the `model-config` file.

xp-model-identifier

The X Print Service allows specification of DPA Printer object attribute definitions across two configuration files: the `attributes/printer` file and the `models/*/model-config` file. The **xp-model-identifier** is defined to provide an association between these two files.

The **xp-model-identifier** attribute value is specified in the `attributes/printer` file. This value corresponds to the name of a model subdirectory under the `models` configuration directory. The X Print Service obtains initial printer attributes from the `model-config` file in the named model subdirectory.

The value consists of the manufacturer and model of the printer. It is recommended that the value consist of only uppercase characters, since either the model identifier or the printer name (typically lowercase) may function as a qualifier for attribute definitions within the configuration files. Valid characters for the value of **xp-model-identifier** are **a**-**z**, **A**-**Z**, **0**-**9**, **_**, and **-**.

Example values are **HPDJ1600C**, **IBM-4039-161**, and **SUN_NP20**.

There is no default value for this attribute. Validation for this attribute is as described for single valued attributes in 7.2.2 above. If a `model-config` file cannot be found based on the value, the value is considered invalid.

The initial value of the **xp-model-identifier** attribute is typically specified by the system administrator in the `printer` attributes file.

xp-page-attributes-supported

A list of page attributes supported for the printer. This list is comprised of a set of whitespace-delimited attribute names.

The value of the **xp-page-attributes-supported** attribute is determined by the print ddx driver.

xp-raw-formats-supported

This attribute identifies the set of data formats recognized as valid values for the *doc_fmt* parameter of the **XpPutDocumentData** function, when this function is called within a print document of *type* **XPDocRaw**. See the "XpStartDoc - XpEndDoc - XpCancelDoc" and "XpPutDocumentData" sections of the "X Print Service Extension Library" chapter for details.

The value is a list of data formats. Each entry in the list is a structure comprised of the data format, an optional format variant, and an optional format version. Structure values are enclosed by curly braces "{}" and delimited by whitespace. Valid values are defined based on the phyiscal printer's capabilities. Examples include **{PostScript 2}** and **{PCL 5}**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation entails syntax checking only.

The initial value of the **xp-raw-formats-supported** attribute is typically set by the printer vendor in the `model-config` file.

xp-setup-proviso

This attribute indicates whether or not a required attribute or set of attributes must be set (typically via user interaction with the Print Dialog Manager) prior to commencing the print job. This attribute will not be utilized by the Sample Implementation print ddx drivers. However, the DtPrintSetupBox will check this attribute along with the **xp-setup-state** job attribute in order to ensure forward compatibility with drivers that may require this (eg. a fax driver that requires a phone number provided by the user).

Valid values for this attribute are **xp-setup-mandatory** and **xp-setup-optional**. If this attribute is not specified, **xp-setup-optional** is assumed.

The initial value of the **xp-setup-proviso** attribute is typically set by the printer vendor in the `model-config` file.

xp-spooler-command

This attribute can be used to override the default spooling operation performed by the X Print Server. The value consists of a command plus any command line options. The resulting print file is passed to the command via stdin.

The command line may contain references to a predefined set of variables, that will be expanded by the server. The variables are:

%printer-name%	the name of the printer
%copy-count%	the value of the **copy-count** attribute
%job-name%	the value of the **job-name** attribute
%options%	the value of the **xp-spooler-command-options** attribute

Example: **/opt/mystuff/bin/my_lp -printer %printer-name%**

The initial value of the **xp-spooler-command** attribute is typically not specified.

Sample Printer Attributes

```
xp-printer-model-identifier: HPLJ4SI
xp-server-list: ganymede.acme.com:6
content-orientations-supported: portrait landscape
descriptor: LaserJet in the West Wing
document-formats-supported: {PostScript 2} {PCL 5}
medium-source-sizes-supported: \
  {top {iso-a4 FALSE {10 200 10 287}} {iso-a5 FALSE {10 138 10 200}} } \
  {bottom {iso-a4 FALSE {10 200 10 287}} {iso-a5 FALSE {10 138 10 200}} }
input-trays-medium: { main iso-a4 } { large-capacity na-letter }
printer-resolutions-supported: 300 600
plexes-supported: simplex duplex tumble
```

7.5 Job Attribute Definitions

7.5.1 Description

Job attributes provide information on how to process a print job. Applications set and retrieve these attributes using **`XpSetAttributes`** and **`XpGetAttributes`** from the X Print Extension API. Typically, job attributes are set by the Print Dialog Manager based on user input from the setup dialog.

The following table shows where querying and / or setting a job attribute value is supported within the X Print Service.

Table 7-1: Job Attribute Usage

Attribute	Configuration	DDX Driver	Application
`job-name`	X	X	X
`job-owner`		X	X
`notification-profile`	X	X	X
`xp-setup-state`	X	X	X
`xp-spooler-command-options`	X	X	X

7.5.2 Job Attributes

`job-name` This is the name of the job to be used in subsequent processing and in printing banner pages. The value is free form text.

No default is provided for this attribute. No validation of the attribute value is performed.

`job-owner` This attribute identifies the human owner of the print job. This attribute is set by **`XpStartJob`** immediately prior to issuing the **`PrintStartJob`** request, and may be used by the **`XpSubmitJob`** driver interface function to identify the user to the spooler. This attribute cannot be set in a configuration file.

`notification-profile`

This attribute is a specification of events about which the user is to be notified. The X Print service uses this attribute to determine whether or not to notify the user of print job completion via electronic mail, or in ISO DPA parlance, the X Print Service recognizes the **`event-report-job-completed`** event with a **`delivery-method`** of **`electronic-mail`**.

Valid values for **`notification-profile`** attribute in the sample implementation are **`{{event-report-job-completed} electronic-mail}`** to send an email message, and **`{}`** if no message is to be sent.

The default value is implicitly taken to indicate that no message be sent. No validation of the attribute value is performed.

`xp-setup-state`

If the value of the **`xp-setup-proviso`** printer attribute is **`xp-setup-manadatory`**, then **`xp-setup-state`** is used to indicate the current setup state as determined by the Print Dialog Manager on behalf of the print ddx driver. If the value of **`xp-setup-proviso`** is **`xp-setup-optional`**, the value of **`xp-setup-state`** is ignored.

Valid values for **`xp-setup-state`** are **`xp-setup-ok`** and **`xp-setup-incomplete`**. **`xp-setup-ok`** indicates that all attributes the ddx driver requires the user to set are valid, indicating a client may commence printing if desired. **`xp-setup-incomplete`** indicates that one or more attributes the driver requires are unspecified or invalid; printing should not be attempted.

The initial value of **`xp-setup-state`** is typically not specified in the `job` attributes configuration file. If **`xp-setup-state`** is unspecified, the default value is **`xp-setup-incomplete`**.

`xp-spooler-command-options`

A free form text string that will be included verbatim on the command line used to invoke the spooler. Valid values are spooler-dependent.

No default is provided for this attribute. No validation of the attribute value is performed.

`xp-spooler-command-results`

A free form text string that will contain the spooler command output that would otherwise appear on a terminal (e.g. stderr and stdout). This text may be useful to present to the user to allow tracking of the resulting spooler job. Applications should retrieve this value following receipt of the **`XPEndJobNotify`** event.

CDEnext

Sample Job Attributes

```
job-name: My Job
xp-spooler-command-options: -onb
```

7.6 Document Attribute Definitions

7.6.1 Description

Document attributes indicate how to process the current document. Applications set and retrieve these attributes using **`XpSetAttributes`** and **`XpGetAttributes`** from the X Print Extension API. Typically, document attributes are set by the DT Print Dialog Manager based on user input from the print dialogs.

The following table shows where querying and / or setting a document attribute value is supported within the X Print Service.

Table 7-1: Document Attribute Usage

Attribute	Configuration	DDX Driver	Application
`content-orientation`	X	X	X
`copy-count`	X	X	X
`default-printer-resolution`	X	X	X
`default-input-tray`	X	X	X
`default-medium`	X	X	X
`document-format`	X	X	X
`plex`	X	X	X
`xp-listfonts-modes`	X	X	X

7.6.2 Document Attributes

`content-orientation`

Specifies the orientation to be used for this document. Valid values are: **`portrait`**, **`landscape`**, **`reverse-portrait`**, and **`reverse-landscape`**.

The default value is implicitly determined by the ddx driver to be the first entry in the value of the **`content-orientations-supported`** printer attribute. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The value must appear in the **`content-orientations-supported`** attribute value to be considered valid.

`copy-count`

Specifies the number of copies of this document to print.

The default value is implicitly taken to be **`1`** by the X Print Server. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The value must be a positive integer.

`default-printer-resolution`

Specifies the resolution in dots per inch to be used for this document.

The default value is implicitly determined by the ddx driver to be the first entry in the value of the **printer-resolutions-supported** printer attribute. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The value must appear in the **printer-resolutions-supported** attribute value to be considered valid.

default-input-tray

The name of the input tray from which media will be drawn for printing the document. Valid values are: **top**, **middle**, **bottom**, **envelope**, **manual**, **large-capacity**, **main**, and **side**. If the **default-medium** attribute is specified, it will take precedence over **default-input-tray**.

No default is assumed for this attribute, since the **default-medium** attribute takes precedence. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The input tray must be included in the **medium-source-sizes-supported** printer attribute value (*note:* if an entry in **medium-source-sizes-supported** omits the input tray specifier, then the input tray value specified for **default-input-tray** will be considered valid, provided of course that it is listed as one of the valid values above).

default-medium

Specifies the medium on which the document is to be printed. The X Print Service defines valid **default-medium** values to be the standard values of the **medium-size** attribute as specified in ISO/IEC 10175-1.

The default value is implicitly determined by the ddx driver, provided the **default-input-tray** attribute is unspecified. The default will correspond to the first medium size found in the value of the **medium-source-sizes-supported** printer attribute. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The value must appear in the **medium-source-sizes-supported** attribute value to be considered valid.

document-format

Specifies the format of the document. The value is a structure comprised of the document-format, an optional document-format-variant, and an optional document-format-version. Specific printer ddx drivers may require specification of the optional values. The structure values are enclosed by curly braces "{}" and delimited by whitespace. Valid values in the sample implementation are **{ PCL 5 }** and **{ PostScript 2 }**.

The default value is determined by the ddx, and is explicitly set in the printer pool. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The value must appear in the **document-formats-supported** printer attribute value to be considered valid.

plex

Specifies the **plex** to be used for this document. Valid values are **simplex**, **duplex**, and **tumble**.

CDEnext

The default value is implicitly determined by the ddx driver to be the first entry in the value of the **plexes-supported** printer attribute. Validation for this attribute is as described for single valued attributes in 7.2.2 above. The value must appear in the **plexes-supported** attribute value to be considered valid.

xp-listfonts-modes

The value of this attribute controls the behavior of the **XListFonts** Xlib function as well as related calls such as **XLoadFont** when these calls are made on a display that has a print context set on it. The value is a whitespace delimited list of one or more listfonts mode values. Valid listfonts mode values in the sample implementation are **xp-list-internal-printer-fonts** and **xp-list-glyph-fonts**.

In the following discussion, references to **XListFonts** should be taken to mean all related Xlib functions that operate on the list of fonts provided by the X Server.

When a print context is set on a display connection, the default behavior of **XListFonts** is to list all of the fonts normally associated with the X print server (i.e. fonts containing glyphs) as well as any internal printer fonts defined for the printer. See the "Fonts" chapter for details on printer fonts. The **xp-listfonts-modes** attribute is provided so that applications can control the behavior of **XListFonts**, typically to show just internal printer fonts. Using only internal printer fonts is useful for performance reasons; the glyphs associated with the font are contained within the printer and do not have to be downloaded to it.

If the value of **xp-listfonts-modes** includes **xp-list-glyph-fonts** **XListFonts** will include all of the fonts available to the server which have glyphs associated with them. If the value of **xp-listfonts-modes** includes **xp-list-internal-printer-fonts** then **XListFonts** will include all of the fonts defined as internal printer fonts.

The default value is implicitly determined by the ddx driver to be the all of the listfonts modes specified in the **xp-listfonts-modes-supported** printer attribute. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above. Each listfonts mode value must appear in the **xp-listfonts-modes-supported** attribute value to be considered valid.

Sample Document Attributes

```
default-printer-resolution: 300
plex: duplex
content-orientation: portrait
document-format: {PCL 5}
copy-count: 2
default-input-tray: side
```

7.7 Page Attribute Definitions

7.7.1 Description

Page attributes are document attributes that can be overridden on a page by page basis within the X Print Service. Applications set and retrieve these attributes using **XpSetAttributes** and **XpGetAttributes** from the X Print Extension API.

The default for each page attribute is the current value of the corresponding document attribute. Validation of page attributes is the same as for document attributes.

The following table shows where querying and / or setting a page attribute value is supported within the X Print Service.

Table 7-1: Page Attribute Usage

Attribute	Configuration	DDX Driver	Application
content-orientation		X	X
default-printer-resolution		X	X
default-input-tray		X	X
default-medium		X	X
plex		X	X
xp-listfonts-modes	X	X	X

CDEnext

7.7.2 Page Attributes

content-orientation

Specifies the orientation to be used for this page. Valid values are: **portrait**, **landscape**, **reverse-portrait**, and **reverse-landscape**.

default-printer-resolution

Specifies the resolution in dots per inch to be used for this page.

default-input-tray

The name of the input tray from which media will be drawn for printing the page. Valid values are: **top**, **middle**, **bottom**, **envelope**, **manual**, **large-capacity**, **main**, and **side**. If the **default-medium** attribute is specified, it will take precedence over **default-input-tray**.

default-medium

Specifies the medium on which the page is to be printed. The X Print Service defines valid **default-medium** values to be the standard values of the **medium-size** attribute as specified in ISO/IEC 10175-1.

plex

Specifies the **plex** to be used for this page. Valid values are **simplex**, **duplex**, and **tumble**.

xp-listfonts-modes

The value of this attribute controls the behavior of the **XListFonts** Xlib function as well as related calls such as **XLoadFont** when these calls are made on a display that has a print context set on it. The value is a whitespace delimited list of one or more listfonts mode values. Valid listfonts mode values in the sample implementation are **xp-list-internal-printer-fonts** and **xp-list-glyph-fonts**.

In the following discussion, references to **XListFonts** should be taken to mean all related Xlib functions that operate on the list of fonts provided by the X Server.

When a print context is set on a display connection, the default behavior of **XListFonts** is to list all of the fonts normally associated with the X print server (i.e. fonts containing glyphs) as well as any internal printer fonts defined for the printer. See the “Fonts” chapter for details on printer fonts. The **xp-listfonts-modes** attribute is provided so that applications can control the behavior of **XListFonts**, typically to show just internal printer fonts. Using only internal printer fonts is useful for performance reasons; the glyphs associated with the font are contained within the printer and do not have to be downloaded to it.

If the value of **xp-listfonts-modes** includes **xp-list-glyph-fonts** **XListFonts** will include all of the fonts available to the server which have glyphs associated with them. If the value of **xp-listfonts-modes** includes **xp-list-internal-printer-fonts** then **XListFonts** will include all of the fonts defined as internal printer fonts.

The default value is implicitly determined by the ddx driver to be the all of the listfonts modes specified in the **xp-listfonts-modes-supported** printer attribute. Validation for this attribute is as described for multi-valued attributes in 7.2.2 above. Each listfonts mode value must appear in the **xp-listfonts-modes-supported** attribute value to be considered valid.

Sample Page Attributes

```
content-orientation: landscape
default-printer-resolution: 150
default-medium: na-legal
plex: simplex
```

7.8 See Also

- “X Print Service Extension Library” chapter.
- “X Print Configuration Databases” chapter.
- “X Print Driver Interface” chapter.
- “Application Print Dialogs” chapter.

FONTS

8.1 Overview

Fonts play an important role in the printing environment. The basic tenet of the DtPrint X Server is to act like a regular X server. X programmers will find the interface familiar.

PURPOSE

Provide the ability to render text.

DESCRIPTION

Fonts may come from several sources:

- Fonts built into the printer (both bitmapped and scalable).
- Bitmapped fonts on the server's local disk.
- Scalable and bitmapped fonts in a format compatible with the printer.
- Fonts from a font server.

From a printing application's point of view, the LoadFont, QueryFont, and ListFonts requests work as usual after the creation and setting of a print context. If the document-formats-supported attribute contains multiple document formats, then the client must set the document-format attribute prior to performing any font requests. All fonts must be on the font path for the print context. In the sample implementation, that font path is identical to the server's font path. That is to say, in the sample implementation there is one server-wide font path. ListFonts returns a list of fonts available along the font path. The X Logical Font Description (XLFD) 1.5 standard is supported.

Font Path Handling. In the sample print server there is one server-wide font path. At server initialization time the font path element corresponding to each printer model configured into the server is added at the front of the server's font path. This means that the font path elements for the printer internal fonts precede the font path elements for other font types. The font renderer for the printer internal font path elements inspects the client performing any font-related request, and responds differently based on whether or not the client has set a print context, and if so, then depending on the model of printer specified in the print context. If the client has not set a print context, or if the client's print context specifies a printer model other than that associated with the particular font path element, then the renderer will not find or return any fonts. If the client has set a print context and the printer specified by that print context matches the model associated with the font path element, then the renderer responds to the font request with information derived from the ".pmf" and other files (e.g. fonts.alias) in the fonts directory within that printer model's configuration directory.

Fonts built into the printer (both bitmapped and scalable). Users will generally prefer to use internal fonts for performance reasons: they already reside in the printer and do not have to be downloaded.The configuration directory for each printer containing internal fonts has a subdirectory named "fonts". This directory contains ".pmf" files defining the metrics for all glyphs in the font. The ".pmf" file format is analogous to that of a ".pcf" file with the glyphs omitted.

PCF bitmapped fonts on the server's local disk. The print server treats these fonts like ordinary X fonts. In response to a LoadFont request, the server will scale the font as required and, in the case of the PCL driver, will convert the font into a format appropriate to the printer and download the font. QueryFont will return an X Font Structure containing metrics for the font.

Fonts from an X font server. These fonts are analogous to having PCF fonts on disk. The difference is that these scale inside the font server.

DEPENDENCIES

Print properties rely on X standard mechanisms:

- Xlib.
- X protocol.
- font server technology.

8.2 Systems Administration Considerations

8.2.1 Related Information

FILES

The print driver's configuration directory stores the metrics for the printer's internal fonts. It contains the font metrics in *pmf* files. The *pmf* files are identical to pcf files, but with glyphs removed. A *fonts.dir* is an index to the fonts. A *fonts.alias* file provides font names consistent with the X Logical Font Description (XLFD) Version 1.5.

X PRINT DRIVER INTERFACE

9.1 Xp Print Driver Overview

9.1.1 PURPOSE

This chapter describes the interfaces used to integrate the print drivers into a server with the Xp extension. This section includes descriptions of the functions a driver is required to implement in order to cooperate with the Xp extension, and descriptions of some utility functions available for the convenience of driver writers. Not covered here are normal DDX driver interfaces for core X functionality.

9.1.2 DESCRIPTION

The X Print server is simply an X server with the Xp extension. The drivers effectively provide a mapping from most X protocol rendering operations to a form understandable by a particular class of printer. The drivers are much like the hardware-specific display drivers in any other X server, but need to have some slightly different and extended capabilities in order to cooperate with the Xp extension, and with the configuration capabilities exposed via the Print Dialog Manager and its associated setup dialogs.

9.1.3 DEPENDENCIES

The print drivers are tightly coupled with the X server itself, and the initial sample print server will be based on the X11-R6 server as supplied by the X Consortium.

9.1.4 ISSUES

9.2 X Print Driver Initialization

9.2.1 Information Available During Initialization

The driver has the following practical sources of information during its initialization:

- Command line arguments - The driver's initialization routine is passed argc and argv corresponding to the arguments passed on the command line to the server.
- Information in the ScreenRec - The driver's initialization routine is passed a pointer to a ScreenRec containing potentially useful information. In particular the width, height, mmWidth, and mmHeight fields are filled in with the maximum potential dimensions prior to the calling of the driver's initialization routine.
- Driver-specific configuration files - The driver can attempt to read information from on-disk files it may expect to be in place on the system.

9.2.2 Xp Extension Initialization Interface

The Xp extension is a bit abnormal relative to other X server extensions. In particular, it is possible to have this extension be applicable on a subset of the screens of a given server. This enables a workstation with an attached printer to utilize a single process for both the X display and the Xp functions. Another somewhat unusual aspect of this extension is that the implementation of its functionality is highly device dependent, and thus each driver must support a set of entry points beyond those provided by normal DDX-compatible drivers. To these ends the driver's initialization routine (i.e. the function which might be called from dix:addScreen) must call a function to provide a pointer to the driver's InitContext function.

9.3 XpRegisterInitFunc

9.3.1 Short Description

Provides the printer-independent print server code with a pointer to the driver's routine to be called when a print context is being initialized for a printer associated with this driver.

9.3.2 Long Description

NAME

XpRegisterInitFunc - register an InitContext function with the device-independent print server code.

SYNOPSIS

```
void XpRegisterInitFunc(ScreenPtr pScreen, int(*InitContext)(),
                        char *driverName);
```

ARGUMENTS

pScreen	Specifies a pointer to a ScreenRec indicating a screen which is prepared to support the Xp extension.
initContext	Specifies a pointer to the function to be called when a print context is initialized.
driverName	Specifies the name of the driver. The names defined in the CDE sample are: XP-RASTER, XP-PCL, and XP-POSTSCRIPT.

RETURN VALUE

None.

DESCRIPTION

The XpRegisterInitFunc provides to the printer-independent portion of the X print server a pointer to the routine to be called during the creation and initialization of a print context associated with a printer which this driver supports.

9.4 Attribute Concepts

Much of the functionality of the Xp system is controlled via the setting of various `attributes`. The attributes both describe the capabilities of the printer, and allow the user and/or the application to control many aspects of the printed output. Most of the attributes are defined in the ISO 10175 and POSIX 1387.4 standards, and are broken into a few different pools.

9.4.1 Server Attributes

These attributes are read-only to the driver. They are created and initialized when the server is initialized, and remain unchanged until the server recycles or is restarted.

9.4.2 Printer Attributes

These attributes are writable only by the print driver. An application can only read these values, as they are a description of the capabilities of the printer and driver combination. These attributes include a description of the available and supported media types, and the supported page description languages among others.

9.4.3 Document Attributes

These attributes describe such things as the media to use for the document, the "plex" to use, and the orientation (i.e. portrait or landscape). These attributes can be read and written by both the application and the driver. Default values for these attributes are set by the driver (possibly using the provided utility routines) when a new print context is initialized. The user or application can modify these attributes to communicate such choices to the driver. It is the driver's responsibility to communicate these attributes to the specific printer, presumably by embedding the appropriate page description language strings in the output. Changes in these attributes may cause the driver to perform operations such as resizing a window referenced by a subsequent **`StartPage`** to fit the specified media size or orientation.

9.4.4 Page Attributes

These are a subset of the document attributes which can be varied on a page-by-page basis. This allows, for example, an application to print a particular page in landscape orientation in the middle of a document which is otherwise in portrait orientation. These attributes can be read and written by both the application and the driver. It is the driver's responsibility to communicate these attributes to the specific printer, typically by embedding the appropriate page description language strings in the output. Changes in these attributes may cause the driver to perform operations such as resizing a window to fit the specified media size or orientation when **`StartPage`** is executed.

9.4.5 Job Attributes

These control the functioning of the spooler itself, allowing the specification of items such as the banner page contents. These attributes can be read and written by both the application and the driver, however the driver should be able to be blissfully unaware of these attributes if the driver chooses to utilize the **`XpSubmitJob`** call documented below. These attributes are ignored if the client specifies the *`save_data`* field to be **`XPGetData`** in its call to **`StartJob`**.

9.5 Attribute Store and Spooler Interface Functions

The functions described in this section are intended as conveniences for the drivers in implementing their **`GetAttribute`**, **`SetAttribute`**, and **`EndJob`** functions. The DDX drivers are *not required* to use the functions described in this section, but it is *strongly recommended* that they do so. These functions provide the driver with insulation from the underlying print spooling system, and are intended to allow drivers developed for the initial sample server to function in environments where more capable printing systems (e.g. Palladium) are in place. These functions do not attempt to mirror the API afforded by the Palladium system, but should provide sufficient capabilities to allow a driver access to all attributes accessible via a Palladium based-system. A driver which chooses not to use these functions is unlikely to integrate smoothly into a Palladium-based environment. Note that the Attribute Store functions do no error checking of Printer, Document, or Page attributes, as such checking is left entirely in the hands of the driver. Driver writers are advised to write their context initialization code such that it gets the attributes, and edits them prior to responding to the first **`GetAttributes`** request from a client. A store of Job attributes is maintained and error-checked internally by the attribute store.

From a driver's perspective the Attribute Store consists of four distinct collections of attributes: Printer Attributes, Document Attributes, Job Attributes, and PageAttributes. All of these attributes are writable for the driver, even though the protocol specifies that the Printer Attributes are read-only. This write access allows the driver to modify the attributes to more accurately describe the capabilities it possesses. As an example, immediately after initialization of the Attribute Store the Printer Attributes may contain an entry stating that the document-formats-supported include both PCL and PostScript (e.g. for a HP-DeskJet 1600C). If the driver only supports a single document-format then the driver should change the document-formats-supported attribute to reflect the fact that it only supports its single document-format. There are separate attribute stores maintained on a per-print-context basis. All strings are in the form accepted by **`XrmGetStringDatabase`**().

9.6 XpInitAttributes

9.6.1 Short Description

Causes the Attribute Store to be initialized. In the initial sample implementation, this causes the Attribute Store to read the initial attribute values from the on-disk configuration files if they have not been read previously. The driver typically calls this routine in the function invoked by **`InitPrintContext`**. The attributes are expected to carry forward unchanged between jobs within the same print context, but the closing and re-initializing of a client's print context should result in freshly initialized attributes. To effect this, a driver should call **`XpInitAttributes`** once and only once for each **`InitPrintContext`** request. After the Attribute Store is initialized for a client, any changes made to the attribute store for the client should remain intact until the print context is destroyed.

9.6.2 Long Description

NAME

XpInitAttributes - initialize the attributes for a particular print context.

SYNOPSIS

```
void XpInitAttributes(PrintContextPtr pContext);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.

RETURN VALUE

None.

DESCRIPTION

`XpInitAttributes` serves to initialize the attribute store associated with a particular print context. It is expected that a driver will call this function upon receipt of an **`InitContext`** request. A driver must call **`XpInitAttributes`** prior to calling either **`XpGetAttributes`**, **`XpGetOneAttribute`**, **`XpAugmentAttributes`** or **`XpSetAttributes`** for a given context.

9.7 XpGetOneAttribute

9.7.1 Short Description

Retrieves from the Attribute Store the current value of a specified attribute.

9.7.2 Long Description

NAME

XpGetOneAttribute - obtain the current value of the specified attribute for a given print context.

SYNOPSIS

```
char *XpGetOneAttribute(PrintContextPtr pContext, XpAttrType pool,
                        char *attributeName);
```

ARGUMENTS

pContext	Specifies a pointer to the print Context for which the attribute value is desired.
pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr, XPServerAttr.
attributeName	Specifies the name of the attribute for which the value is desired.

RETURN VALUE

A pointer to a character string containing the value of the specified attribute, or NULL if the attribute does not exist in the attribute store. The returned string must not be freed.

DESCRIPTION

The XpGetOneAttribute function returns a string containing the value of the specified attribute as a string.

CDEnext

9.8 XpGetAttributes

9.8.1 Short Description

Retrieves from the Attribute Store the current contents of the specified set of attributes in its entirety.

9.8.2 Long Description

NAME

XpGetAttributes - obtain the current contents of the specified attribute set for a given print context.

SYNOPSIS

```
char *XpGetAttributes(PrintContextPtr pContext, XpAttrType pool);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr, XPServerAttr.

RETURN VALUE

A pointer to a character string containing the current set of attributes for the print context. It is the caller's responsibility to free the string when it is no longer needed.

DESCRIPTION

The XpGetAttributes function returns a string containing the current attribute names and values. It is expected that drivers will use this function in order to implement the GetAttributes function.

9.9 XpGetMediumDimensions

9.9.1 Short Description

Retrieves from the Attribute Store the dimensions of the medium currently selected for the document associated with a particular print context.

9.9.2 Long Description

NAME

XpGetMediumDimensions - obtain the dimensions of the medium for a document associated with a particular print context.

SYNOPSIS

```
void XpGetMediumDimensions(PrintContextPtr pContext, CARD16 *width,
   CARD16 *height);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
width	Returns the width of the medium.
height	Returns the height of the medium.

RETURN VALUE

None.

DESCRIPTION

The XpGetMediumDimensions function provides a convenient means for the driver to determine the overall dimensions of the medium specified for the current (or first) page of the document in the job associated with the specified print context. The mediumDimensions returned are computed from the value of the **`default-medium`** attribute, or if it is not specified, from the **`default-input-tray`** and **`input-trays-medium`** attributes. If neither of these attribute sets is valid, then XpGetMediumDimensions returns values corresponding to the first entry in the list of **`medium-source-sizes-supported`**. The returned dimensions are in pixel units, through the use of the **`content-orientation`** and **`default-resolution`** document attributes.

CDEnext

9.10 XpGetReproductionArea

9.10.1 Short Description

Retrieves from the Attribute Store the net reproducible area for the document associated with a particular print context.

9.10.2 Long Description

NAME

XpGetReproductionArea - obtain the dimensions and position of the reproducible area for a document associated with a particular print context.

SYNOPSIS

```
void XpGetReproductionArea(PrintContextPtr pContext, xRectangle *pRect);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
pRect	Specifies a pointer to a rectangle which will return the reproducible area.

RETURN VALUE

None.

DESCRIPTION

The XpGetReproductionArea function provides a convenient means for the driver to determine the dimensions of the reproducible area for the current (or first) page of the document in the job associated with the specified print context. The reproducible area differs from the medium dimensions in that the reproducible area has had subtracted from it any regions of the medium which cannot be printed on, and all regions which the printer mechanism cannot mark. The returned dimensions are in units of pixels, through the use of the **`content-orientation`** and **`default-resolution`** document attributes. The relevant medium is determined from the contents of either the **`default-medium`** attribute, or if that is not defined the **`default-input-tray`** and **`input-trays-medium`** attributes. The **`medium-source-sizes-supported`** attribute is used to determine the reproducible area for this medium. If insufficient information is available from the attributes then the values returned will correspond to North American Letter media with a one-quarter-inch non-reproducible border.

9.11 XpAugmentAttributes

CDEnext

9.11.1 Short Description

Augments the values of the specified attribute class.

9.11.2 Long Description

NAME

XpAugmentAttributes - augment the value of a particular attribute class for a given print context.

SYNOPSIS

```
void XpAugmentAttributes(PrintContextPtr pContext, XpAttrType pool,
                          char *attributes);
```

ARGUMENTS

`pContext`	Specifies a pointer to the print context for which the attributes are desired.
`pool`	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr.
`attributes`	Specifies the names and values of the attributes.

RETURN VALUE

None.

DESCRIPTION

The XpAugmentAttributes function adds the supplied attributes to the specified attribute class. If a supplied attribute already exists, then its new value is taken from the supplied list of attributes.

9.12 XpSetAttributes

9.12.1 Short Description

Stores a new set of attributes for a particular class of attributes.

9.12.2 Long Description

NAME

XpSetAttributes - set new attributes and values for a given print context.

SYNOPSIS

```
void XpSetAttributes(PrintContextPtr pContext, XpAttrType pool, char
   *attributes);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are to be set.
pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr, XPServerAttr.
attributes	A string of all the attributes and values for the specified class.

RETURN VALUE

None.

DESCRIPTION

The XpSetPrintAttributes function accepts a string containing the new attribute names and values. It is expected that drivers will use this function in order to implement the SetAttributes function.

9.13 XpSubmitJob

9.13.1 Short Description

Requests that a particular job file be submitted to the spooler with an associated set of job attributes.

CDEnext

9.13.2 Long Description

NAME

XpSubmitJob - submit a file to the print spooler.

SYNOPSIS

```
void XpSubmitJob(char *fileName, PrintContextPtr pContext);
```

ARGUMENTS

fileName	Specifies the name of the file to be submitted for printing.
pContext	Specifies the print context associated with the print job.

RETURN VALUE

None.

DESCRIPTION

XpSubmitJob takes whatever steps are necessary to submit the specified file to the underlying spooling system with the specified job attributes. It is expected that drivers will call this function from within their EndJob functions. In the initial sample implementation this function invokes the lp command to spool the job.

9.14 XpFreeAttributes

9.14.1 Short Description

Frees the storage associated with the attributes for a specified XpContext.

9.14.2 Long Description

NAME

XpFreeAttributes - free the memory associated with the attributes for a particular XpContext.

SYNOPSIS

```
void XpFreeAttributes(PrintContextPtr pContext);
```

ARGUMENTS

pContext	Specifies the print context associated with the print job.

RETURN VALUE

None.

DESCRIPTION

XpFreeAttributes frees the memory associated with the attributes for the specified print context. **XpFreeAttributes** should be called from the driver's **DestroyContext** function.

9.15 Xp Extension Functions

A print driver *must* implement the following set of functions which provide the underpinnings for the extension requests defined by the Xp extension. The InitContext call is the function which was passed to XpRegisterInitFunc, while the other functions are called via function pointers stored in each PrintContext. A pointer to a PrintContext is passed to each of these routines, and has the following structure:

```
typedef struct _xpprintfuncs {
int versionNumber;
int (*StartJob)(); /* pPrintContext, saveData */
int (*EndJob)(); /* pPrintContext, cancel */
int (*StartDoc)(); /* pPrintContext */
int (*EndDoc)(); /* pPrintContext, cancel */
int (*StartPage)(); /* pPrintContext, pWin */
int (*EndPage)(); /* pPrintContext, pWin, cancel */
int (*PutDocumentData)(); /* pPrintContext, pWin, pData,len_data, pFmt,
 pOpt */
int (*GetDocumentData)(); /* pPrintContext, client, maxBufferSize */
int (*DestroyContext)(); /* pPrintContext */
char *(*GetAttributes)(); /* pPrintContext, class */
char *(*GetOneAttribute)(); /* pPrintContext, class, attribute */
int (*SetAttributes)(); /* pPrintContext, class, pData */
int (*AugmentAttributes)(); /* pPrintContext, class, pData */
int (*GetMediumDimensions(); /* pPrintContext, pWidth, pHeight */
int (*GetReproducibleArea(); /* pPrintContext, pRect */
} XpDriverFuncs, *XpDriverFuncsPtr;

typedef struct _XpContext {
        XID contextID;
        char *printerName;
        int screenNum;
        struct _XpClient *clientHead; /* list of clients */
        CARD32 state;
        VisualID pageWin;
        DevUnion *devPrivates;
        XpDriverFuncs funcs;
} XpContextRec, *XpContextPtr;
```

When the driver's InitContext function is called it is free to inspect the printerName field of the XpContext, and is required to fill in all of the function pointers in the embedded XpDriversFuncs structure.

CDEnext

9.16 InitContext

9.16.1 Short Description

Provides pointers to functions implementing the various printing related operations for the specified context.

9.16.2 Long Description

NAME

InitContext - initialize the contents of the supplied XpContext.

SYNOPSIS

```
int InitContext(PrintContextPtr pContext);
```

ARGUMENTS

pContext	Specifies a pointer to the print context in which the print data will be generated.

RETURN VALUE

`Success`, or a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`InitContext`** function supplies the driver with the name of the printer to be used in subsequent print jobs in the specified print context. The driver is expected to fill in the function pointers within the XpContext, and to initialize the attribute store for the print context at the time this function is called. This enables an application to then query the printer attributes and receive accurate information. The driver should also initialize any per-context data it wishes to maintain.

9.17 DestroyContext

9.17.1 Short Description

Notifies the driver that the context is no longer in use, and any associated data should be freed.

9.17.2 Long Description

NAME

DestroyPrintContext - release any driver resources allocated for the specified print context.

SYNOPSIS

```
int DestroyContext(PrintContextPtr pContext);
```

ARGUMENTS

pContext	Specifies a pointer to the print context which is being destroyed.

RETURN VALUE

`Success`, or a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`DestroyContext`** function provides the driver an opportunity to clean up any state or resources it has allocated in support of the specified print context. **`XpFreeAttributes`** should be called from this function if the attribute storage facilities have been used to create the attributes store for this context.

CDEnext

9.18 StartJob

9.18.1 Short Description

Implements the driver level functionality of the XpStartJob extension request.

9.18.2 Long Description

NAME

StartJob - begin a new print job associated with a particular window.

SYNOPSIS

```
int StartJob(PrintContextPtr pContext, XPSaveData sendData);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the print job is starting.
sendData	Specifies whether the resulting print data is to be sent to a client, and if so, the driver must be prepared to call XpWriteClientData when there is print output data available to be sent.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`StartJob`** function will typically check for and delete any previously created print data associated with the print context, and will create storage space for the new print job. The *`sendData`* parameter indicates that a client will receive the data, rather than having the data submitted to the spooling system. The driver is then required to call XpSendClientData() when there is print data available. The driver may assume that there will be no changes to the Job attributes for this context after the StartJob function has been called.

9.19 EndJob

9.19.1 Short Description

Implements the driver level functionality of the XpEndJob extension request.

9.19.2 Long Description

NAME

EndJob - Ends the print job associated with a particular window, and submits the job to the printer.

SYNOPSIS

```
int EndJob(PrintContextPtr pContext, Boolean cancel);
```

ARGUMENTS

`pContext`	Specifies a pointer to the print context for which the print job is ending.
`cancel`	A TRUE value indicates that the job is to be canceled, and any remaining print data discarded rather than submitted to the spooler or returned to the client.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`EndJob`** function typically submits the job to the spooler. If *`cancel`* is TRUE then any remaining print data is discarded, and if necessary the print job is canceled. If print data has been sent to a client via XpSendClientData(), then XpSendClientData should be called with the “status” parameter set to either END or CANCEL, depending on the value of the *`cancel`* flag. At that point the driver should be able to properly accept a StartJob request on the same print context.

9.20 StartDoc

9.20.1 Short Description

Implements the driver level functionality of the XpStartDoc extension request.

9.20.2 Long Description

NAME

StartDoc - begins a new document within the print job associated with a window.

SYNOPSIS

```
int StartDoc(PrintContextPtr pContext, XPDocumentType type);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which a new document is starting.
type	Specifies the type of the document. The value is one of: XPDocRaw, XPDocNormal. A value of XPDocRaw indicates that the data is to be passed through unmodified by the print server, and only PutDocumentData calls will be accepted after such a StartDoc.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`StartDoc`** function is primarily a place holder for any necessary functionality needed when and if the Xp Service is implemented on top of a print spooling system which supports multiple documents in a job, such as one compliant with POSIX 1387.4. The driver is guaranteed to receive a **`StartDoc`** call with *`type`* equal to **`XpDocNormal`** prior to receiving a **`StartPage`**. If the driver receives a **`StartDoc`** call with *`type`* equal to **`XpDocRaw`** it can assume it will not receive a **`StartPage`** prior to the **`EndDoc`** for that document. The driver may assume that there will be no changes to the document attributes for the specified context after this function has been called.

9.21 EndDoc

9.21.1 Short Description

Implements the driver level functionality of the XpEndDoc extension request.

CDEnext

9.21.2 Long Description

NAME

EndDoc - ends a document within the print job associated with a print context.

SYNOPSIS

```
int EndDoc(PrintContextPtr pContext, Boolean cancel);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which a document is ending.
cancel	Indicates whether the current document is to be canceled, and any remaining (i.e. buffered) data for this document discarded.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`EndDoc`** function is essentially a place holder for any necessary functionality needed when and if the Xp Service is implemented on top of a print spooling system capable of supporting multiple documents in a single job, such as one compliant with POSIX 1387.4. A driver is guaranteed to receive an **`EndDoc`** prior to an **`EndJob`**.

9.22 StartPage

9.22.1 Short Description

Implements the driver level functionality of the XpStartPage extension request.

9.22.2 Long Description

NAME

StartPage - begins a new page within the print job, and associate the print context with a window.

SYNOPSIS

```
int StartPage(PrintContextPtr pContext, Window pWin);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which a new page is starting.
pWin	Specifies a pointer to the window to be used as the top-most window in the printed page.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`StartPage`** function discards any previously created data for any previous page, allocates any storage which may be necessary for a new page, resizes the window to match the size of the medium, clears the window and all descendent windows to their backgrounds, and adds any necessary page header data to the contents of this page. Such header data is generally determined by the values of the page attributes. The driver may assume that there will be no changes to the Page atttributes for the specified context after this call.

9.23 EndPage

9.23.1 Short Description

Implements the driver level functionality of the XpEndPage extension request.

9.23.2 Long Description

NAME

EndPage - ends a page within the print job associated with a window.

SYNOPSIS

```
int EndPage(PrintContextPtr pContext, Window pWin, Boolean cancel);
```

ARGUMENTS

`pContext`	Specifies a pointer to the print context for which a new page is starting.
`pWin`	Specifies a pointer to the top-most window for the page.
`cancel`	A value of TRUE indicates that any remaining page data should be discarded rather than being submitted as part of the current document.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`EndPage`** function adds any necessary trailing information for the page, and adds the page data to the print job associated with the print context. The trailer data is determined by the values of the page attributes. If *`cancel`* is TRUE, then any buffered page data should be discarded rather than being included in the current document and job.

9.24 PutDocumentData

9.24.1 Short Description

Implements the driver level functionality of the XpPutDocumentData extension request.

9.24.2 Long Description

NAME

PutDocumentData - adds application supplied data to the print document associated with a print context.

SYNOPSIS

```
int PutDocumentData(
    PrintContextPtr pContext;
    Window pWin;
    char *pData;
    int len_data;
    char *pFmt,
    char *pOpt);
```


ARGUMENTS

pContext	Specifies a pointer to the print context defining the print job.
pWin	Specifies a pointer to the window into which the data is to be placed.
pData	Points to the data to be added to the print job.
len_data	Specifies the length in bytes of the data to be added to the print job.
pFmt	Points to a string describing the format of the data (e.g. PCL5).
pOpt	Points to a string describing driver-specific options for the data.

RETURN VALUE

`Success` if no errors are encountered, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The **`PutDocumentData`** function provides a means for an application to supply printer device dependent data of its own creation. The data is added to the document associated with the specified print context. The driver may, if it desires, modify or interpret the data based on the specified format, options, and known printer characteristics. As an example, a driver may choose to support DeviceData formats other than those which are supported by the printer itself by translating the data into a format understood by the printer. If the **`PutDocumentData`** is sent following a **`StartDoc`**(*`printContext,`* **`XPDocNormal)`**, then the driver is expected to provide any generally needed page description language header data necessary to embed the supplied data within the boundaries of the specified window, however, if the **`PutDocumentData`** is sent after a **`StartDoc`**(*`printContext`*, **`XPDocRaw`**), then the driver is expected to pass the data straight through to the spooler with no additions or modifications.

The window given to the PutDocumentData function specifies the size and location of the embedded data. It may not be possible for the driver to clip the embedded data to take into account other windows which occlude the given window.

9.25 GetDocumentData

9.25.1 Short Description

Informs the driver of which client should receive the generated document data for the print job associated with the specified print context.

9.25.2 Long Description

NAME

GetDocumentData - establish which client should receive data generated by print jobs in a print context.

SYNOPSIS

```
int GetDocumentData(PrintContextPtr pContext, ClientPtr client, int
   maxBufferSize);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the print data is desired.
client	Specifies the client which is to receive all generated document data for the job associated with the specified print context.
maxBufferSize	Specifies the maximum amount of data the client wishes to receive in a single reply.

RETURN VALUE

Success if the driver was able to set its state in preparation for returning the document data, else a code indicating the problem (e.g. BadAlloc).

DESCRIPTION

The GetDocumentData function allows the driver to prepare for sending document data for a job to the specified client. If the receiving client is unable to read back the generated data quickly enough to keep up with the rate of data generation the driver is free to suspend the processing of further requests from clients making rendering requests within a print context.

CDEnext

9.26 GetAttributes

9.26.1 Short Description

Returns the current contents of the specified set of attributes.

9.26.2 Long Description

NAME

GetAttributes - obtain the current contents of the specified attribute set for a given print context.

SYNOPSIS

```
char *GetAttributes(PrintContextPtr pContext, XpAttrType pool);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr, XPServerAttr.

RETURN VALUE

A pointer to a character string containing the current set of attributes for the print context. It is the caller's responsibility to free the string when it is no longer needed. GetAttributes returns a NULL pointer in the case of an allocation error (i.e. **BadAlloc**), and returns a pointer to an empty string if the requested attribute store is empty.

DESCRIPTION

The GetAttributes function returns a string containing the current attribute names and values for the specified attribute class. It is expected that drivers will use the **XpGetAttributes** function to implement this function.

9.27 GetOneAttribute

9.27.1 Short Description

Returns the value of the specified attribute within a particular attribute pool for a print context.

9.27.2 Long Description

NAME

GetOneAttribute - obtain the current value of a particular attribute.

SYNOPSIS

```
char *GetOneAttributes(PrintContextPtr pContext, XpAttrType pool, char
    *attr);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr, XPServerAttr.
attr	Specifies the attribute for which the value is desired.

RETURN VALUE

A pointer to a character string containing the value of the attribute for the print context. The caller must not free the returned string. GetOneAttribute returns a NULL pointer in the case of an allocation error (i.e. **BadAlloc**), and returns a pointer to an empty string if the requested attribute is not defined.

DESCRIPTION

The GetOneAttribute function returns a string containing the values for the specified attribute class and attribute within the specified print context. It is expected that drivers will use the **XpGetOneAttribute** function to implement this function.

9.28 AugmentAttributes

9.28.1 Short Description

Augments the contents of the specified set of attributes.

9.28.2 Long Description

NAME

AugmentAttributes - augment the contents of the specified attribute set for a given print context.

SYNOPSIS

```
int AugmentAttributes(PrintContextPtr pContext, XpAttrType pool, char
    *attributes);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are to be augmented.

pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr.
attributes	Specifies the names and values of some attributes for the above-specified class.

RETURN VALUE

`Success` if no error is detected, otherwise a value indicating the error (e.g. **`BadAlloc`**, **`BadAttribute`**).

DESCRIPTION

The **`AugmentAttributes`** function adds the specified attributes to the store of the specified attribute class. If a supplied attribute already exists in the store, then the value supplied in this call will become the value of that attribute.

9.29 SetAttributes

9.29.1 Short Description

Sets the contents of the specified set of attributes.

9.29.2 Long Description

NAME

SetAttributes - set the contents of the specified attribute set for a given print context.

SYNOPSIS

```
int SetAttributes(PrintContextPtr pContext, XpAttrType pool, char
   *attributes);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
pool	Specifies the pool of the attribute which is desired. This is one of XPJobAttr, XPDocAttr, XPPrinterAttr, XPPageAttr.
attributes	Specifies the names and values of all the attributes for the above-specified class.

RETURN VALUE

`Success` if no error is detected, otherwise a value indicating the error (e.g. **`BadAlloc`**, **`BadAttribute`**).

DESCRIPTION

The SetAttributes function replaces the existing attributes and values (if any) with those contained in the *attributes*. It is expected that drivers will use the **XpSetAttributes** function to implement this function.

9.30 Xp Utility and Convenience Functions

The functions described in this section are intended as conveniences for the drivers.

9.31 XpSendData

9.31.1 Short Description

Send printer data to any client which has performed an XpGetDocumentData call for a specific print context.

9.31.2 Long Description

NAME

XpSendData - send print data to any interested client.

SYNOPSIS

```
int XpSendData(PrintContextPtr pContext, ClientPtr client, char *data,
    int len_data);
```

ARGUMENTS

pContext	Specifies a pointer to the print context for which the attributes are desired.
client	A pointer to the client which is to receive the data.
data	A pointer to the print data to be sent to the interested client.
len_data	Specifies the length in bytes of the print data.

RETURN VALUE

Success if no error is detected, otherwise a value indicating the error (e.g. **BadAlloc**, **BadAttribute**).

DESCRIPTION

The XpSendData function sends the supplied data to the specified client. The client should be that which has performed an XpGetDocumentData call for the specific print context. The returned value indicates any error which occurred during the sending of the data. This function takes care of formatting the data into GetDocumentDataReply structures including byte-swapping reply header information.

9.32 XpAllocateContextPrivateIndex

9.32.1 Short Description

Allocate a context private index for use by the driver.

9.32.2 Long Description

NAME

XpAllocateContextPrivateIndex - allocate a context private index for use by the driver.

SYNOPSIS

```
int XpAllocateContextPrivateIndex();
```

ARGUMENTS

RETURN VALUE

An index value which can be used in a subsequent call to XpAllocateContextPrivate.

DESCRIPTION

The XpAllocateContextPrivateIndex function returns an index into the context devPrivates array for use by the caller. This index may be passed to XpAllocateContextPrivate to have the printer-independent portion of the server automatically allocate a fixed amount of memory with each context.

9.33 XpAllocateContextPrivate

9.33.1 Short Description

Inform the printer-independent code of the amount of memory to be allocated with each context for use by the caller.

9.33.2 Long Description

NAME

XpAllocateContextPrivate - allocate an amount of memory with each context for use by the caller.

SYNOPSIS

```
int XpAllocateContextPrivate(int index, int amount);
```

ARGUMENTS

index	Specifies an index returned by XpAllocateContextPrivateIndex.
amount	The amount of memory to be allocated with each context for use by the caller.

RETURN VALUE

`Success` if no error is detected, otherwise a value indicating the error (e.g. **`BadAlloc`**).

DESCRIPTION

The XpAllocateContextPrivate function informs the printer-independent portion of the server how much memory to allocate with each context for the use of the caller.

X Print Extension Protocol

10.1 Protocol Overview

10.1.1 PURPOSE

The following describes the X Print Extension Protocol. The X Print Extension Protocol concentrates on print job and page management. It also includes provisions for applications to pass device-specific data to the printer. The X Print Extension Protocol works only on screens that support the X Print Extension.

10.1.2 DEPENDENCIES

The X Print Extension is an extension to the Core X protocol, and cannot be used outside of the X environment.

10.1.3 ISSUES

10.2 Request Protocol Specifications

10.2.1 PrintQueryVersion

Errors: none.

This request returns the version information for the Xp extension.

Encoding: *PrintQueryVersion Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	0	minor opcode
2	1	request length

CDEnext

Encoding: *PrintQueryVersion Reply*

Number of Bytes	Value or Type	Description
1	1	Reply
1	unused	
2	CARD16	sequence number
4	0	reply length
2	CARD16	major version
2	CARD16	minor version
20	unused	

10.2.2 PrintGetPrinterList

Errors: BadAlloc

Returns an array of structures describing the printers accessible through this server.

Encoding: *PrintGetPrinterList Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	1	minor opcode
2	3+(nl+np + ll+lp)/4	request length
4	CARD32	printerNameLen
4	CARD32	localeLen
nl	STRING8	printerName
np	BYTE	p=pad(nl)
ll	STRING8	locale
lp	BYTE	lp=pad(ll)

Encoding: *PrintGetPrinterList Reply*

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequenceNumber
4	(8 + nl+nlp + dl+dlp)/4 computed listCount times	length
4	CARD32	listCount
20		unused
(8 + nl+nlp + dl+dlp) computed listCount times	LISTofPRINTER	list of printers

PRINTER		
4	CARD32	nameLen
nl	STRING8	name
nlp	BYTE	nlp=pad(nl)
4	CARD32	descLen
dl	STRING8	desc
dlp	BYTE	dlp=pad(dl)

10.2.3 PrintRehashPrinterList

Errors: none.

Causes the X-Server to recompute the list of available (active/valid) printers.

Encoding: *PrintRehashPrinterList Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	20	minor opcode

Encoding: PrintRehashPrinterList Request

Number of Bytes	Value or Type	Description
2	1	request length

10.2.4 PrintCreateContext

Errors: BadMatch

Specifies which printer is to be used for any subsequent jobs started from this client connection, and associates a print context ID with this job/printer combination. The screen defines the possible values for the printer name.

Encoding: PrintCreateContext Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	2	minor opcode
2	4 + (nl+np + ll+lp)/4	request length
4	CARD32	contextID
4	CARD32	printerNameLen
4	CARD32	localeLen
nl	STRING8	printerName
np	BYTE)	np=pad(nl)
ll	STRING8	locale
lp	BYTE	lp=pad(ll)

10.2.5 PrintSetContext

Errors: XPBadContext

Specifies the print context to be used in any print-related calls from this client. This call is typically used by a client to join in cooperatively rendering a print job.

Encoding: *PrintSetContext Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	3	minor opcode
2	2	request length
4	CARD32	printContext

10.2.6 PrintGetContext

Errors: none.

Retrieve the currently set print context.

Encoding: *PrintGetContext Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	4	minor opcode
2	1	request length

Encoding: *PrintGetContext Reply*

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
4	CARD32	printContext
16		unused

CDEnext

10.2.7 PrintDestroyContext

Errors: XPBadContext

Disassociates the client from its current print context.

Encoding: *PrintDestoryContext Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	5	minor opcode
2	2	request length
4	CARD32	printContext

10.2.8 GetContextScreen

Errors: XPBadContext

Encoding: *GetContextScreen Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	6	minor opcode
2	1	request length

Encoding: *GetContextScreen Reply*

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length

Encoding: *GetContextScreen Reply*

Number of Bytes	Value or Type	Description
4	WINDOW	rootWindow
16		unused

10.2.9 PrintStartJob

Errors: XPBadContext, XPBadSequence, BadValue

The server takes whatever actions are necessary to define a new print job, including discarding any previously accumulated print data for the specified print context. This request results in the generation of a XpNotify event with the detail field set to `Xp_StartJobEvent`. If the save-data flag is TRUE, then the server will store the print data, will generate XP_DataAvailable events when there is print data readable by the client, and will not submit the resulting job to the printer.

Encoding: *PrintStartJob Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	7	minor opcode
2	2	request length
1	CARD8	saveData
3		unused

10.2.10 PrintEndJob

Errors: XPBadContext, XPBadSequence

The print job associated with the print context ends, and if the cancel flag is false, the accumulated print data is sent to the printer (or an Xp_data_available event is generated). This request results in the generation of a XpNotify event with the detail field set to `Xp_EndJobEvent`.

Encoding: PrintEndJob Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	8	minor opcode
2	2	request length
1	BOOL	cancel
3		unused

10.2.11 PrintStartDoc

Errors: XPBadContext, XPBadSequence, BadValue

The server takes whatever actions are necessary to define a new document. This request results in the generation of a XpNotify event with the detail field set to `Xp_StartDocEvent`.

Encoding: PrintStartDoc Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	9	minor opcode
2	2	request length
1	CARD8	type
3		unused

10.2.12 PrintEndDoc

Errors: XPBadContext, XPBadSequence

The current document associated with the print context ends. This request results in the generation of a XpNotify event with the detail field set to `Xp_EndJobEvent`.

Encoding: PrintEndDoc Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	10	minor opcode
2	2	request length
1	BOOL	cancel
3		unused

10.2.13 PrintPutDocumentData

Errors: XPBadContext, XPBadSequence, BadValue, BadDrawable, XPBadResourceID

The supplied data is added to the contents of the current job.

Encoding: PrintPutDocumentData Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	11	minor opcode
2	4 + (d+dp + f+fp + o+op)/4	request length
4	DRAWABLE	drawable
4	CARD32	len_data
2	CARD16	len_fmt
2	CARD16	len_options
d	LISTofBYTE	data
dp	BYTE	dp=pad(d)
f	STRING8	doc_fmt
fp	BYTE	fp=pad(f)
o	STRING8	options
op	BYTE	op=pad(o)

CDEnext

10.2.14 PrintGetDocumentData

Errors: XPBadContext, XPBadSequence

Returns accumulated print data. Will return no more that max-bytes number of bytes of data.The returned bytes-remaining specifies the number of bytes remaining to be read as of the time of processing the **`XpGetDocumentData`** request.

Encoding: *PrintGetDocumentData Request*

Number of Bytes	Value	Description
1	base	major opcode
1	12	minor opcode
2	3	request length
4	PCONTEXT	printContext
4	CARD32	maxBufferSize

Encoding: *PrintGetDocumentData Reply sent multiple times*

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequence number
4	(n + p)/4	reply length
4	0 XPGetDocFinished 1 XPGetDocSecondConsumer	statusCode
4	CARD32	finishedFlag
4	CARD32	dataLen
12		unused
n	LISTofBYTE	data
p	BYTE	p=pad(n)

10.2.15 PrintStartPage

Errors: XPBadContext, XPBadSequence, BadWindow, XPBadResourceID, BadValue

Window is configured, it and all of its children are cleared to their background, and Expose events are sent to all affected windows. This request results in the generation of a XpNotify event with the detail field set to `Xp_StartPageEvent` following any and all of the configure and expose events.

Encoding: *PrintStartPage Request*

Number of Bytes	Value or Type	Description
1	base?	major opcode
1	13	minor opcode
2	2	request length
4	WINDOW	window

10.2.16 PrintEndPage

Errors: XPBadContext, XPBadSequence

The rendering defining the page's content ends, and if the cancel flag is false then the current page contents are added to the contents of the job associated with the print context. This request results in the generation of a XpNotify event with the detail field set to `Xp_EndPageEvent`.

Encoding: *PrintEndPage Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	14	minor opcode
2	2	request length
1	BOOL	cancel
3		unused

10.2.17 PrintSelectInput

Errors: XPBadContext, BadValue

Specifies which print events the client is interested in.

Encoding: PrintSelectInput Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	15	minor opcode
2	3	request length
4	PCONTEXT	printContext
4	BITMASK	eventMask

10.2.18 PrintInputSelected

Errors: XPBadContext

Encoding: PrintInputSelected Request

Number of Bytes	Value	Description
1	base	major opcode
1	16	minor opcode
2	2	request length
4	PCONTEXT	printContext

Encoding: PrintInputSelected Reply

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
4	BITMASK	eventMask
4	BITMASK	allEventsMask
16		unused

10.2.19 PrintGetAttributes

Errors: XPBadContext, BadValue, BadAlloc

`XpGetAttributes` returns the name-value pairs which comprise the list of attributes in the specified attribute class.

Encoding: *PrintGetAttributes Request*

Number of Bytes	Value	Description
1	base	major opcode
1	17	minor opcode
2	3	request length
4	PCONTEXT	printContext
1	CARD8	type
3		unused

Encoding: *PrintGetAttributes Reply*

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequence number
4	(n+p)/4	reply length
4	CARD32	stringLen
20		unused
n	STRING8	string
p		p=pad(n)

10.2.20 PrintGetOneAttribute

Errors: XPBadContext, BadValue, BadAlloc

CDEnext

Encoding: *PrintGetOneAttribute Request*

Number of Bytes	Value	Description
1	base	major opcode
1	19	minor opcode
2	4 + (n+p)/4	request length
4	PCONTEXT	printContext
4	CARD32	nameLen
1	CARD8	type
3		unused
n	STRING8	name
p		p=pad(n)

Encoding: *PrintGetOneAttribute Reply*

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused
2	CARD16	sequence number
4	(n+p)/4	reply length
4	CARD32	valueLen
20		unused
n	STRING8	value
p		p=pad(n)

10.2.21 PrintSetAttributes

Errors: XPBadContext, XPBadSequence, BadValue, BadMatch, BadAlloc

`XpSetAttributes` sets the names and values for all attributes within the specified attribute class.

Encoding: *PrintSetAttributes Request*

Number of Bytes	Value	Description
1	base	major opcode
1	18	minor opcode
2	4 + (n+p)/4	request length
4	PCONTEXT	printContext
4	CARD32	stringLen
1	CARD8	type
1	CARD8	rule
2		unused
n	STRING8	string
p	BYTE	p=pad(n)

10.2.22 PrintGetPageDimensions

Errors: XPBadContext

Retrieve dimensions related to the currently select media.

Encoding: *PrintGetPageDimensions Request*

Number of Bytes	Value or Type	Description
1	base	major opcode
1	21	minor opcode
2	2	request length
4	PCONTEXT	print context

Encoding: *PrintGetPageDimensions Reply*

Number of Bytes	Value or Type	Description
1	1	Reply

Encoding: PrintGetPageDimensions Reply

Number of Bytes	Value or Type	Description
1		unused
2	CARD16	sequence number
4	0	reply length
2	CARD16	width
2	CARD16	height
2	CARD16	rx
2	CARD16	ry
2	CARD16	rwidth
2	CARD16	rheight
12		unused

10.2.23 PrintQueryScreens

Errors: none

Retrieve a list of screens supporting the Xp Extension.

Encoding: PrintQueryScreens Request

Number of Bytes	Value or Type	Description
1	base	major opcode
1	22	minor opcode
2	2	request length

Encoding: PrintQueryScreens Reply

Number of Bytes	Value or Type	Description
1	1	Reply
1		unused

Encoding: *PrintQueryScreens Reply*

Number of Bytes	Value or Type	Description
2	CARD16	sequence number
4	listCount	reply length
4	CARD32	listCount
20		unused
4 * listCount	LISTofROOTWINDOW	list of root windows

ROOTWINDOW		
4	WINDOW	rootWindow

10.3 Event Protocol Specifications

10.3.1 PrintNotify

Encoding: `PrintNotify`

Number of Bytes	Type or Value	Description
1	0 + base	code
1	0 XPStartJobNotify 1 XPEndJobNotify 2 XPStartDocNotify 3 XPEndDocNotify 4 XPStartPageNotify 5 XPEndPageNotify	detail
2	CARD16	sequence number
4	PCONTEXT	print context
1	BOOL	cancel flag
23		unused

CDEnext

10.3.2 AttributeNotify

Encoding: `AttributeNotify`

Number of Bytes	Type or Value	Description
1	1 + base	code
1	1 XPJobAttr 2 XPDocAttr 3 XPPageAttr 4 XPPrinterAttr 5 XPServerAttr 6 XPMediumAttr (future use) 7 XPSpoolerAttr (future use)	detail
2	CARD16	sequence number
4	PCONTEXT	print context
24		unused

10.4 Error Protocol Specifications

10.4.1 BadContext

This error is generated whenever one of the Xp extension requests is made with an invalid printer context ID.

Encoding: *BadContext*

Number of Bytes	Value or Type	Description
1	0	Error
1	0 + base	code
2	CARD16	sequence number

10.4.2 BadSequence

This error is generated when a Xp request is made out of sequence. This will happen if, for instance, a StartPage request is recieved prior to the receipt of a StartJob request.

Encoding: *BadSequence encoding*

Number of Bytes	Value or Type	Description
1	0	Error
1	1 + base	code
2	CARD16	sequence number

10.4.3 BadResourceID

This error is generated when an XID (X-Resource) is valid in general, not not valid when used as an X-Resource in the X Print Extension. Most often, the error is generated when an XID is created outside the applicable print context or on the wrong screen.

Encoding: *BadResourceID encoding*

Number of Bytes	Value or Type	Description
1	0	Error
1	2 + base	code
2	CARD16	sequence number

EXAMPLES

FILES

SEE ALSO

APPENDIX A Application Print Dialogs

10.5 Introduction

From the user's perspective in a typical application, printing is done through a series of dialogs, the first one being initiated by selecting, for example, the pulldown menu <File><Print...>. This chapter documents a set of dialogs provided by CDEnext for use primarily by applications that perform X printing. The initial print dialog is provided as the **`DtPrintSetupBox`** widget, and includes two convenience functions for creating the widget: **`DtCreatePrintSetupBox`** and **`DtCreatePrintSetupDialog`**. A dialog for selecting X Printers may be invoked by **`DtPrintSetupBox`**. This dialog is known as the **`DtPrinterSelectionDialog`**. A dialog for obtaining additional printer information, the **`DtPrinterInfoDialog`**, may be invoked from either the **`DtPrintSetupBox`** or the **`DtPrinterSelectionDialog`**. All of these dialogs are considered part of the **`DtPrintSetupBox`** widget, and as such, no external API exists for these dialogs.

Although **`DtPrintSetupBox`** is designed primarily for X printing, it is also designed for use as a general application print dialog for use in any CDEnext application that provides printing.

The remaining sections of this chapter document the CDEnext application print dialogs and convenience functions:

- "DtPrintSetupBox"
- "DtCreatePrintSetupBox"
- "DtCreatePrintSetupDialog"
- "DtPrintFillSetupData"
- "DtPrintCopySetupData"
- "DtPrintFreeSetupData"
- "DtPrintResetConnection"
- "DtPrintSetupProc"
- "DtPrinterSelectionDialog" (includes **`DtPrinterInfoDialog`**)

10.6 DtPrintSetupBox

10.6.1 Short Description

`DtPrintSetupBox` is a widget that is typically the initial window used to set various options prior to printing from an application. This widget is primarily designed for use by applications that utilize the X Print Service. However, the widget interface is also designed to be flexible enough for use by applications employing other printing methods.

10.6.2 Long Description

NAME

DtPrintSetupBox - Application Print Setup Widget

SYNOPSIS

```
#include <Dt/Print.h>
```

DESCRIPTION

`DtPrintSetupBox` is a widget that is typically the initial window used to set various options prior to printing from an application. This widget is primarily designed for use by applications that utilize the X Print Service. However, the widget interface is also designed to be flexible enough for use by applications employing other printing methods.

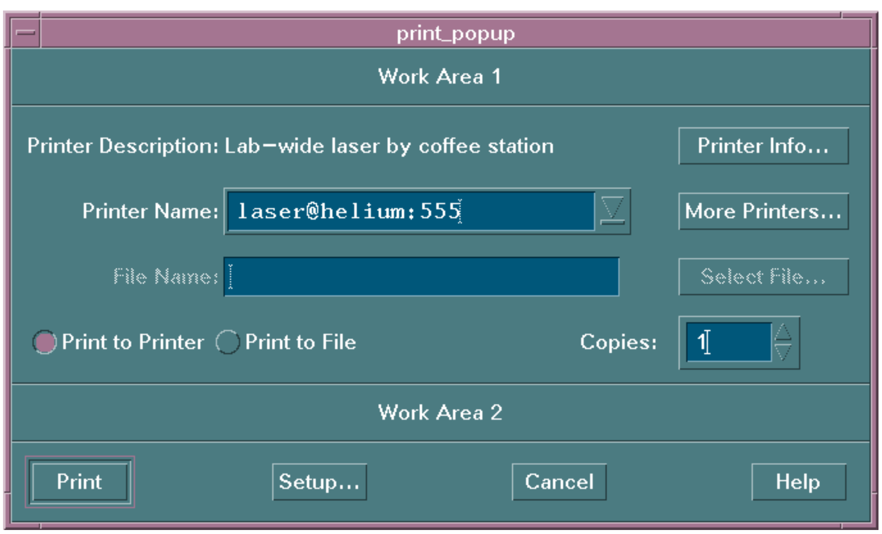


Figure 10-1. DtPrintSetupBox - Application Print Setup Widget

The **DtPrintSetupBox** is organized based on generic print options and application specific print options. The sections are clearly demarcated with separators to set off the generic section from the application specific section(s). By utilizing the **DtNworkAreaLocation** resource, the application developer may choose to utilize an area above the generic section, below the generic section, or both above and below the generic section.

The four default buttons (Print, Setup, Cancel, and Help) are considered generic buttons. Applications may create additional pushbuttons as children of **DtPrintSetupBox**. These buttons will be laid out following the Print button.

The Printer Name combo box contains the X Printer Specifier of the printer to be used for the print job. The X Printer Specifier is an identifier that uniquely identifies an X Printer. The format of this specifier is *printerName*@*host*:*display*.

Descendants

DtPrintSetupBox creates the descendents shown in the following table. An application can use **XtNameToWidget** to gain access to the named descendent. In addition, a user or an application can use the descendent name when specifying resource values.

Table 10-1: DtPrintSetupBox Descendents

Named Descendent	Class	Identity
BottomWorkAreaSeparator	**XmSeparatorGadget**	Separator above the bottom work area
ButtonSeparator	**XmSeparatorGadget**	Separator above the pushbuttons
Cancel	**XmPushButtonGadget**	Cancel button
Copies	**XmSimpleSpinBox**	SpinBox containing the number of copies
CopiesLabel	**XmLabelGadget**	Label for the Copies SpinBox
Description	**XmLabelGadget**	Printer description
DestFile	**XmToggleButton**	Print to file radio button
DestPrinter	**XmToggleButton**	Print to printer radio button
DestRadioBox	**XmRowColumn**	Manager for print to printer and print to file radio buttons
FileName	**XmTextField**	File name field
FileNameLabel	**XmLabelGadget**	Label for the file name field
Help	**XmPushButtonGadget**	Help button
Info	**XmPushButtonGadget**	Printer information button
Name	**XmComboBox**	Printer name
NameLabel	**XmLabelGadget**	Label for the printer name field
Print	**XmPushButtonGadget**	Print button

CDEnext

Table 10-1: DtPrintSetupBox Descendents

Named Descendent	Class	Identity
SelectFile	**XmPushButtonGadget**	Select File button
SelectPrinter	**XmPushButtonGadget**	Select Printer button
Setup	**XmPushButtonGadget**	Setup button
TopWorkAreaSeparator	**XmSeparatorGadget**	Separator below the top work area

Classes

The DtPrintSetupBox inherits behavior and resources from the **Core**, **Composite**, **Constraint**, **XmManager**, and **XmBulletinBoard** superclasses.

The class pointer is **dtPrintSetupBoxWidgetClass**.

The class name is **DtPrintSetupBox**.

New Resources

Table 10-2: DtPrintSetupBox Resources

Name	Class/Type	Access	Default Value
DtNcancelCallback	**DtCCancelCallback/ XtCallbackList**	C	**NULL**
DtNclosePrintDisplayCall-back	**DtCCloseDisplayCall-back/XtCallbackList**	C	**NULL**
DtNcopies	**DtCCopies/int**	CSG	1
DtNdescription	**DtCDescription/ XmString**	CSG	dynamic
DtNfileName	**DtCPrintToFileName/ String**	CSG	**NULL**
DtNminimizeButtons	**DtCminimizeButtons/ Boolean**	CSG	**False**
DtNoptionCount	**DtCOptionCount/ Cardinal**	CSG	0
DtNoptions	**DtCOptions/ XtPointer**	CSG	**NULL**
DtNprintCallback	**DtCPrintCallback/ XtCallbackList**	C	**NULL**
DtNprintDestination	**DtCPrintDestination/ XtEnum**	CSG	**DtPRINT_TO_PRINTER**

Table 10-2: DtPrintSetupBox Resources

Name	Class/Type	Access	Default Value
DtNprinterInfoProc	`DtCPrinterInfoProc`/ `DtPrintSetupProc`	CSG	dynamic
DtNprinterName	`DtCPrinter`/`String`	CSG	dynamic
DtNprintSetupMode	`DtCPrintSetupMode`/ `XtEnum`	CG	`DtPRINT_SETUP_XP`
DtNselectFileProc	`DtCSelectFileProc`/ `DtPrintSetupProc`	CSG	default procedure
DtNselectPrinterProc	`DtCSelectPrinterProc`/ `DtPrintSetupProc`	CSG	dynamic
DtNsetupCallback	`DtCSetupCallback`/ `XtCallbackList`	C	`NULL`
DtNverifyPrinterProc	`DtCVerifyPrinterProc`/ `DtPrintSetupProc`	CSG	dynamic
DtNworkAreaLocation	`DtCworkAreaLocation`/ `XtEnum`	CSG	`DtWORK_AREA_BOTTOM`

DtNcancelCallback

Specifies the list of callbacks that is called when the Cancel button is activated. The callback reason is `DtPRINT_CR_CANCEL`.

DtNclosePrintDisplayCallback

When the value of the **DtNprintSetupMode** resource is `DtPRINT_SETUP_XP`, the `DtPrintSetupBox` will manage the X printing display connection and print context. As such, **DtNclosePrintDisplayCallback** is provided to allow an application to perform any desired processing (such as destroying windows created on the print display) before the `DtPrintSetupBox` destroys the current print context and closes the current print display connection.

This callback list will not be called if the value of the **DtNprintSetupMode** resource is anything other than `DtPRINT_SETUP_XP`.

The callback reason is `DtPRINT_CR_CLOSE_PRINT_DISPLAY`.

DtNcopies

The number of copies of the document to print. This is a spin box into which the user may enter a positive integer.

DtNdescription

A description of the printer as provided by the system administrator.

DtNfileName

Specifies the name of the destination file. Setting this resource will update the value of the File Name text field.

DtNminimizeButtons

If false, sets the dimensions of all of the buttons at the bottom of the widget to the width of the widest button and the height of the tallest button. If true, the dimensions of the buttons are not altered.

DtNoptionCount

The number of **XrmOptionDescRec** records specified in **DtNoptions**. This value is passed as the *num_options* argument to **XtDisplayInitialize** when the **DtPrintSetupBox** establishes a connection to an X Print Server

If the value of the **DtNprintSetupMode** resource is anything other than **DtPRINT_SETUP_XP**, this resource is ignored.

DtNoptions

A pointer to a list of **XrmOptionDescRec** records that the **DtPrintSetupBox** will pass unaltered as the *options* argument to **XtDisplayInitialize** when the **DtPrintSetupBox** establishes a connection to an X Print Server. The **DtPrintSetupBox** does not make an internal copy of the storage pointed to by this resource. It is the caller's responsibility to ensure that the value of this resource points to a valid storage location or **NULL** for the lifetime of the widget instance.

If the value of the **DtNprintSetupMode** resource is anything other than **DtPRINT_SETUP_XP**, this resource is ignored.

DtNprintCallback

Specifies the list of callbacks that is called when the Print button is activated. The callback reason is **DtPRINT_CR_PRINT**. This callback is used to initiate the print job.

DtNprintDestination

Indicates where the print output should be directed. Valid values for this resource are:

DtPRINT_TO_FILE

Direct print output to a file. The destination file name is indicated by the **DtNfileName** resource. Setting this value will cause the Print to File radio button to be selected, enable the File Name label and text field, and enable the Select File button.

DtPRINT_TO_PRINTER

Direct print output to a printer. The destination printer is indicated by the **DtNprinterName** resource. Setting this value will cause the Print to Printer radio button to be selected, disable the File Name label and text field, and disable the Select File button.

DtNprinterInfoProc

This resource specifies the procedure that will be used to present printer information in response to activation of the Printer Information button. The printer selection dialog presented by the default **DtNselectPrinterProc** will also call this procedure in response to activation of its Printer Information button. If the value of the **DtNprintSetupMode** resource is **DtPRINT_SETUP_XP**, a default procedure that presents a printer information dialog is used. For other values of **DtNprintSetupMode**, the default value of the **DtNprinterInfoProc** is **NULL**.

This procedure typically does not update either **DtPrintSetupBox** resources or X Print Service attributes.

The return value of this procedure is ignored by **DtPrintSetupBox**. However, it is recommended that the procedure follow the conventions presented in the "DtPrintSetupProc" section to ensure future compatibility.

DtNprinterName

The name of the printer to send the print job to.

If the value of the **DtNprintSetupMode** resource is **DtPRINT_SETUP_XP**, setting this resource will update the Printer Name field based on the value of the **XpPrinterNameMode** XRM resource. See the "EXTERNAL INFLUENCES" section below for more information.

If the value of the **DtNprintSetupMode** resource is **DtPRINT_SETUP_PLAIN**, setting this resource will update the value of the Printer Name text field with the value of this resource.

DtNprintSetupMode

Instructs the widget as to whether or not it is being used in an application that utilizes the X Print Service. If so, then the widget will manage the X printing display connection and print context, and provide defaults for a number of X printing operations, such as printer selection and information dialogs, and printer verification. Refer to individual resource descriptions to determine if and how they are affected by the value of this resource.

Valid values for this resource are:

DtPRINT_SETUP_PLAIN

This widget will be used by an application that performs its own print document format generation and print job submission.

DtPRINT_SETUP_XP

This widget will be used by an application that utilizes the X Print Service to perform print document format generation and print job submission.

DtNselectFileProc

This resource specifies the procedure that will be used in response to activation of the Select File button. The default value for this resource is a pointer to a procedure which will invoke an **XmFileSelectionBox** dialog to select a file name. If the user cancels the file selection dialog no **DtPrintSetupBox** components will be updated. If the user selects a file name, the file name will be set as the value for the **DtNfileName** resource.

This procedure communicates the newly selected file name to the **DtPrintSetupBox** by setting the **DtNfileName** resource. Since the default procedure presents a File Selection Dialog, the **DtNfileName** resource is actually set after the procedure returns, in response to the user activating the File Selection Dialog's Ok pushbutton.

The return value of this procedure is ignored by **DtPrintSetupBox**. However, it is recommended that the procedure follow the conventions presented in the "DtPrintSetupProc" section to ensure future compatibility.

DtNselectPrinterProc

This resource specifies the procedure that will be used in response to activation of the Select Printer button. If the value of the **DtNprintSetupMode** resource is **`DtPRINT_SETUP_XP`**, a default procedure that invokes a **`DtPrinterSelectionDialog`** is used. If the user cancels the printer selection dialog no **`DtPrintSetupBox`** components will be updated. If the user selects a printer, the printer will be set as the value for the **DtNprinterName** resource.

This procedure communicates the newly selected printer name to the **`DtPrintSetupBox`** by setting the **DtNprinterName** resource. Since the default procedure presents a Printer Selection Dialog, the **DtNprinterName** resource is actually set after the procedure returns, in response to the user activating the Printer Selection Dialog's Ok pushbutton.

If the value of the **DtNprintSetupMode** resource is anything other than **`DtPRINT_SETUP_XP`**, the default value of the **DtNselectPrinterProc** is **`NULL`**.

The return value of this procedure is ignored by **`DtPrintSetupBox`**. However, it is recommended that the procedure follow the conventions presented in the "DtPrintSetupProc" section to ensure future compatibility.

DtNsetupCallback

Specifies the list of callbacks that is called when the Setup button is activated. The callback reason is **`DtPRINT_CR_SETUP`**. This callback is used to initiate a detailed setup dialog, such as the Print Dialog Manager.

DtNverifyPrinterProc

This resource specifies the procedure that will be used to verify the current value of the **DtNprinterName** resource before any operation requiring a valid printer is performed. If the current value of the **DtNprinterName** resource is **`NULL`**, this procedure will set a default printer as the value of the **DtNprinterName** resource.

If this procedure provides a default printer name, or a fully qualified X printer name, it should communicate the new name to the **`DtPrintSetupBox`** by setting the **DtNprinterName** resource before returning.

If the value of the **DtNprintSetupMode** resource is **`DtPRINT_SETUP_XP`**, a default procedure will be set as the value of the **DtNverifyPrinterProc** resource. This default procedure will verify the X printer, open a print display connection by calling **`XOpenDisplay`**, create a print context by calling **`XpCreateContext`**, and set the print context by calling **`XpSetContext`**. This procedure does not call **`XtDisplayInitialize`** for the new display connection; the **`DtPrintSetupBox`** is responsible for Xt display initialization. The procedure communicates the new print display and context to the **`DtPrintSetupBox`** by updating the *`print_data->print_display`* and *`print_data->print_context`* elements of the callback structure prior to returning.

If the value of the **DtNprintSetupMode** resource is anything other than **`DtPRINT_SETUP_XP`**, the default value of the **DtNverifyPrinterProc** is **`NULL`**.

If the value of the **DtNverifyPrinterProc** resource is **`NULL`**, the printer name is always considered valid.

If this procedure determines the printer name is valid or sets a valid printer name (and X printer connection information), it should return **DtPRINT_SUCCESS**. If the printer name is invalid or no valid default can be determined, this procedure should return **DtPRINT_FAILURE**.

DtNworkAreaLocation

Indicates how to position work area children within the **DtPrintSetupBox**. Possible values are:

DtWORK_AREA_BOTTOM

A single work area child may be added, and will be placed below the generic controls and above the pushbuttons at the bottom of the window. A managed separator will be placed between the work area and the generic controls. This is the default.

DtWORK_AREA_TOP

A single work area child may be added, and will be placed above the generic controls and below the top of the window. A managed separator will be placed between the work area and the generic controls.

DtWORK_AREA_TOP_AND_BOTTOM

Two work area children may be added. The first work area created will become the top work area, positioned with a separator as for **DtWORK_AREA_TOP**, and the second will become the bottom work area, positioned with a separator as for **DtWORK_AREA_BOTTOM**.

The effect of adding more work area children than indicated by the value of **DtNworkAreaLocation** is undefined.

CDEnext

Inherited Resources

The **DtPrintSetupBox** inherits resources from the **XmBulletinBoard**, **XmManager**, **Constraint**, **Composite**, and **Core** superclasses. Please refer to the reference pages for these superclasses for inherited resources and their descriptions.

The **DtPrintSetupBox** redefines the default value of the **XmBulletinBoard** resource **XmNnoResize** as **True**.

Callback And Procedure Resource Information

DtPrintSetupBox defines a new structure, **DtPrintSetupData**, that is passed to callbacks and procedure resource values. For callbacks only, **DtPrintSetupBox** defines a new callback structure, **DtPrintSetupCallbackStruct**. Not all fields in these structures are valid for all callbacks and procedures. For callbacks, the application must first look at the *reason* field, and use only the structure members that are valid for that particular reason. For each procedure, the application should only reference structure members that are defined as valid for that particular procedure. The **DtPrintSetupData** and **DtPrintSetupCallbackStruct** structures are defined as follows:

```
typedef struct
{
    String          printer_name;
    Display         *print_display;
    XPContext       print_context;
```

```
    XtEnum              destination;
    String              dest_info;
} DtPrintSetupData;
```

printer_name Contains the current value of the **DtNprinterName** resource.

print_display If **DtNprintSetupMode** is **DtPRINT_SETUP_XP**, *print_display* contains a pointer to the **Display** structure for the current X Printer. **XtDisplayInitialize** is guaranteed to have been called for *print_display* prior to the **DtPrintSetupBox** calling any callback. For other values of **DtNprintSetupMode**, this field is **NULL**.

print_context If **DtNprintSetupMode** is **DtPRINT_SETUP_XP**, *print_context* contains the print context handle for the current X Printer. **XpSetContext** is guaranteed to have been called for *print_context* on *print_display* prior to the **DtPrintSetupBox** calling any callback. For other values of **DtNprintSetupMode**, this field is **NULL**.

destination Contains the current value of the **DtNprintDestination** resource.

dest_info Additional information about the print destination as indicated by the *destination* field.

If *destination* is **DtPRINT_TO_FILE** this field contains the name of the file to print to.

If *destination* is **DtPRINT_TO_PRINTER** this field contains the name of the currently selected printer as determined by the current value of the **XpPrinterNameMode** resource. This is useful for display within dialogs displaying print status, etc. because it is the printer name as presented to the user in the **DtPrintSetupBox**.

```
typedef struct
{
    int                 reason;
    XEvent              *event;
    DtPrintSetupData    *print_data;
} DtPrintSetupCallbackStruct;
```

reason Indicates why the callback was invoked.

event Points to the **XEvent** that triggered the callback. It can be **NULL**.

print_data Points to a **DtPrintSetupData** structure containing additional callback information.

The following table indicates for each callback reason, which **DtPrintSetupCallbackStruct** and **DtPrintSetupData** members are valid:

Table 10-3: Valid Fields By Callback Reason

Reason	Valid Fields
DtPRINT_CR_CANCEL	*reason, event*
DtPRINT_CR_CLOSE_PRINT_DISPLAY	*reason, printer_name, print_display, print_context*
DtPRINT_CR_PRINT	*reason, event, printer_name, print_display, print_context, destination, dest_info*
DtPRINT_CR_SETUP	*reason, event, printer_name, print_display, print_context*

The following table indicates for each procedure resource, which **DtPrintSetupData** members are valid:

Table 10-4: Valid Fields By Procedure Resource Name

Procedure	Valid Fields
DtNprinterInfoProc	*printer_name, print_display, print_context*
DtNselectFileProc	*destination, dest_info*
DtNselectPrinterProc	*printer_name*
DtNverifyPrinterProc	*printer_name, print_display, print_context*

Translations

DtPrintSetupBox inherits translations from **XmBulletinBoard**.

Virtual Bindings

The bindings for virtual keys are implementation-dependent. For information about bindings for virtual buttons and keys, see VirtualBindings(3X).

EXTERNAL INFLUENCES

The following specifies application resources and environment variables that will influence the behavior of the **DtPrintSetupBox**. If a given resource is defined, it will have precedence over the corresponding environment variable. There is no corresponding environment variable for the **XpPrinterNameMode** resource.

XRM Application Resources

XpPrinter

This variable defines the default destination X Printer Specifier for the **DtPrintSetupBox**. If the specifier is just a *printerName*, the *host*:*display* portion of the specifier is obtained by checking if the X Server to which the client application is connected to is an X Print Server managing *printerName*, otherwise the first *server* in the **XpServerList** or XPSERVERLIST that manages the printer will be used. If the `:`*display* number is omitted, `:0` is assumed.

Example: `Dtmail*XpPrinter: laser_1@callisto:6`

XpPrinterNameMode

This resource indicates how an X Printer Specifier shall be shown in the Printer Name combo box text. Valid values for this resource are:

`DtSHORT_NAME`
: Display only the *printerName* portion of the X Printer Specifier.

`DtMEDIUM_NAME`
: Display the printer name as a combination of the *printerName* and the *host* portions of the X Printer Specifier with an intervening “at” (`@`) symbol. For example “printer@host”.

`DtLONG_NAME`
: Display the fully qualified X Printer Specifier. For example “printer@host:6”

If this resource is not specified, **DtPrintSetupBox** will assume a default of `DtSHORT_NAME`.

XpPrinterList

This resource defines the initial set of X Printer Specifiers shown in the Printer Name combo box list.

The resource value is a whitespace-delimited list of partially or fully specified X Printer Specifiers. When the user selects a specifier from this list, if the specifier is just a *printerName*, the *host*:*display* portion of the specifier is obtained by checking if the X Server to which the client application is connected to is an X Print Server managing *printerName*, otherwise the first *server* in the **XpServerList** or XPSERVERLIST that manages the printer will be used. If the `:`*display* number is omitted, `:0` is assumed.

Example:

```
*xpPrinterList: laser laser2@argon:3 laser7@xenon
```

XpServerList

This resource contains a list of X Print Server specifiers. Each entry in the list is of the form *host*`:`*display*, and is separated from other entries by whitespace. **DtPrintSetupBox** uses this list to fully qualify partial X Printer Specifiers consisting of just the *printerName*.

Example: `*.XpServerList: hanz:6 franz:6 ahnold:6`

Environment Variables

PDPRINTER, LPDEST, PRINTER
: If the XPRINTER environment variable and the **XpPrinter** resource are not specified, the **`DtPrintSetupBox`** will check the environment variables (in order) PDPRINTER, LPDEST, and PRINTER to obtain a *printerName* which can be used to generate an X Printer Specifier to use for the default X Printer shown in the Printer Name combo box text field. The *host*:*display* portion of the specifier is obtained by checking if the X Server to which the client application is connected to is an X Print Server managing *printerName*. If not, the list of X Print Servers specified in the **XpServerList** or XPSERVERLIST is queried, until the first X Printer with a matching *printerName* is found.

XPRINTER
: The specification of the XPRINTER environment variable is the same as the **XpPrinter** resource.

XPRINTERLIST
: The specification of the XPRINTERLIST environment variable is the same as the **XpPrinterList** resource.

XPSERVERLIST
: The specification of the XPSERVERLIST environment variable is the same as the **XpServerList** resource.

EXAMPLES

Sample code can be found in the `/proj/cde/examples/dtprint` directory.

SEE ALSO

- The “DtPrintSetupProc”, “DtCreatePrintSetupDialog”, “DtPrintSetupProc”, and “DtPrinterSelectionDialog” sections in this chapter.
- The “X Print Service Extension Library” and “Dt Print Dialog Manager” chapters.

10.7 DtCreatePrintSetupBox

10.7.1 Short Description

DtCreatePrintSetupBox is a convenience function that creates an unmanaged instance of a **DtPrintSetupBox** widget, and returns its widget ID.

10.7.2 Long Description

NAME

DtCreatePrintSetupBox - convenience function to create an instance of a **DtPrintSetupBox** widget.

SYNOPSIS

```
#include <Dt/Print.h>

Widget DtCreatePrintSetupBox(
   Widget              parent,
   const String        name,
   ArgList             arglist,
   Cardinal            argcount)
```

ARGUMENTS

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in *arglist*.

RETURN VALUE

Returns the **DtPrintSetupBox** widget ID.

DESCRIPTION

DtCreatePrintSetupBox is a convenience function that creates an unmanaged instance of a **DtPrintSetupBox** widget, and returns its widget ID.

RELATED INFORMATION

See also “DtPrintSetupBox”, “DtCreatePrintSetupDialog”.

10.8 DtCreatePrintSetupDialog

10.8.1 Short Description

DtCreatePrintSetupDialog is a convenience function that creates an instance of a dialog containing a **DtPrintSetupBox** widget, and returns the **DtPrintSetupBox** widget ID.

10.8.2 Long Description

NAME

DtCreatePrintSetupDialog - convenience function to create a **DtPrintSetupBox** dialog.

SYNOPSIS

```
#include <Dt/Print.h>

Widget DtCreatePrintSetupDialog(
   Widget             parent,
   const String       name,
   ArgList            arglist,
   Cardinal           argcount)
```

ARGUMENTS

parent	Specifies the parent widget ID.
name	Specifies the name of the created widget.
arglist	Specifies the argument list.
argcount	Specifies the number of attribute/value pairs in *arglist*.

RETURN VALUE

Returns the **DtPrintSetupBox** widget ID.

DESCRIPTION

DtCreatePrintSetupDialog is a convenience function that creates a **DialogShell** and an unmanaged **DtPrintSetupBox** child of the **DialogShell**. Use **XtManageChild** to pop up the print set up dialog (passing the **DtPrintSetupBox** as the *widget* parameter); use **XtUnmanageChild** to pop it down.

RELATED INFORMATION

See also “DtPrintSetupBox”, “DtPrintSetupProc”.

10.9 DtPrintFillSetupData

10.9.1 Short Description

DtPrintFillSetupData is used to obtain an X printer connection in order to initiate an X printing job in situations other than direct interaction with a **DtPrintSetupBox** (e.g. a “quick print” button on a toolbar). This printer connection information can be obtained from an existing **DtPrintSetupBox** widget instance, or if a **DtPrintSetupBox** widget instance is unavailable, **DtPrintFillSetupData** will provide a new X printer connection.

10.9.2 Long Description

NAME

DtPrintFillSetupData - obtain X printer connection information

SYNOPSIS

```
#include <Dt/Print.h>

XtEnum DtPrintFillSetupData(
                    Widget wid,
                    DtPrintSetupData *print_data)
```

ARGUMENTS

wid The widget ID of a **DtPrintSetupBox**, or **NULL** if no **DtPrintSetupBox** is available.

print_data A pointer to an existing **DtPrintSetupData** structure that **DtPrintFillSetupData** will update with valid X printer connection information. See the **DtPrintSetupBox** manual page for the definition of the **DtPrintSetupData** structure.

RETURN VALUE

DtPRINT_SUCCESS
The X printer connection was successfully obtained.

DtPRINT_FAILURE
The X printer connection could not be established. The specific reason has been reported by the **DtPrintSetupBox** to the user.

DtPRINT_INVALID_DISPLAY
The indicated X print server could not be found.

DtPRINT_NOT_XP_DISPLAY
The indicated X server does not support the X Printing Extension.

DTPRINT_NO_DEFAULT
A default printer could not be determined.

DtPRINT_NO_PRINTER
The indicated printer could not be found on the X print server, or a default printer could not be determined.

DtPRINT_BAD_PARM
The value passed for *print_data* is **NULL**, or the the value of the **DtNprintSetupMode** resource for *wid* is not **DtPRINT_SETUP_XP**.

DESCRIPTION

DtPrintFillSetupData is used to obtain an X printer connection in order to initiate an X printing job in situations other than direct interaction with a **DtPrintSetupBox** (e.g. a “quick print” button on a toolbar, or “GUI-less” printing). This X printer connection information can be obtained from an existing **DtPrintSetupBox** widget instance, or if a **DtPrintSetupBox** widget instance is unavailable, **DtPrintFillSetupData** will provide a new X printer connection.

A print job is typically initiated when the user selects the “Print” button from within a **DtPrintSetupBox** widget instance. Applications may wish to provide additional avenues for the user to initiate a print job, namely a “Quick Print” toolbar button, or a command line parameter to allow “GUI-less” printing, when a video display is not available. **DtPrintFillSetupData** is designed to provide an X printer connection in a manner consistent with **DtPrintSetupBox**.

For both the “Quick Print” and “GUI-less” cases, the caller may set the *printer_name* element of the passed *print_data* in order to specify the destination X printer. **DtPrintFillSetupData** will treat the passed *printer_name* as if the user had typed the printer name into the Printer Name text field of the **DtPrintSetupBox**. The *printer_name* will be verified, and an X printer connection will be established. The returned *printer_name* may be a fully qualified version of the passed *printer_name* as a result of the printer verification process. For a description of this process, see the **DtNverifyPrinterProc** resource description in the **DtPrintSetupBox** manual page. If **DtPrintFillSetupData** decides to return a new *printer_name*, it will free the passed *printer_name* by calling **XtFree**.

For the “Quick Print” button case, **DtPrintFillSetupData** is intended to be used with an existing **DtPrintSetupBox** instance, so that the “Quick Print” button behaves as if the user had brought up a **DtPrintSetupBox**, selected a printer as indicated by *printer_name,* and activated the “Print” button. The following conditions are particular to this case:

- The *wid* parameter should be set to the widget ID of the **DtPrintSetupBox** instance. **DtPrintFillSetupData** will fill the passed *print_data* structure similarly to how the **DtPrintSetupBox** fills out the *print_data* element of the **DtPrintSetupCallbackStruct** passed as call data to the **DtNprintCallback** list. The only exception is that when using **DtPrintFillSetupData** it is the caller’s responsibility to free the allocated memory locations pointed to by the *print_data* structure by calling **DtPrintFreeSetupData**.
- The X printer connection returned is managed by the **DtPrintSetupBox**. The caller should not destroy the print context, nor close the print display connection.
- It is not required for the **DtPrintSetupBox** widget instance passed via *wid* to have ever been managed prior to calling **DtPrintFillSetupData**.

- If the passed *printer_name* is **NULL**, the current printer as indicated by the **DtNprinterName** resource will be used, and returned in *printer_name*. If the passed *printer_name* is a different printer than indicated by the current value of the **DtNprinterName** resource, the resource will be updated.
- The *destination* and *dest_info* elements of the passed *print_data* will be set according to the current state of the passed **DtPrintSetupBox**. If *dest_info* is set when passed to **DtPrintFillSetupBox**, **DtPrintFillSetupBox** will free the memory by calling **XtFree** if it decides to update *dest_info*.
- If a connection cannot be established, the DtPrintSetupBox will automatically be managed, displaying an error message box. After dismissing the message box, the user can select a new printer and restart the print operation, if desired.

For the "GUI-less" case, **DtPrintFillSetupData** is intended to provide an X printer connection, in a manner consistent with an X printer connection established by **DtPrintSetupBox**, without actually creating a **DtPrintSetupBox**. The following conditions are particular to this case:

- The passed *wid* is set to **NULL**.
- The X printer connection returned is managed by the caller, which means it is up to the caller to destroy the print context (**XpDestroyContext**), and close the print display connection (**XCloseDisplay**). Additionally the print display connection will not be initialized via **XtInitializeDisplay**, as is the case if a **DtPrintSetupBox** widget Id is passed to **DtPrintFillSetupData**.
- If **NULL** is passed for *printer_name*, **DtPrintFillSetupData** will determine a default printer, using the same procedure as **DtPrintSetupBox**, except that XRM resources are not available due to the lack of a video display connection, and set the *printer_name* field to this default printer name upon return.
- The *destination* and *dest_info* elements of the passed *print_data* will be ignored by **DtPrintFillSetupData**.
- The caller can free the allocated memory locations pointed to by the returned *print_data* structure by calling **DtPrintFreeSetupData**.

EXAMPLES

Sample code can be found in the `/proj/cde/examples/dtprint` directory.

RELATED INFORMATION

See the "DtPrintSetupBox", "DtPrintCopySetupData", and "DtPrintFreeSetupData" sections in this chapter.

10.10 DtPrintCopySetupData

10.10.1 Short Description

Copies one **DtPrintSetupData** structure to another.

10.10.2 Long Description

NAME

DtPrintCopySetupData - copy one **DtPrintSetupData** structure to another

SYNOPSIS

```
#include <Dt/Print.h>

DtPrintSetupData* DtPrintCopySetupData(
                              DtPrintSetupData *target,
                              const DtPrintSetupData *source)
```

ARGUMENTS

target	A pointer to the **DtPrintSetupData** structure to copy to.
source	A pointer to the **DtPrintSetupData** structure to copy from.

RETURN VALUE

The *target* pointer.

DESCRIPTION

DtPrintCopySetupData is used to copy the **DtPrintSetupData** structure pointed to by *source* to the **DtPrintSetupData** structure pointed to by *target*. Elements in *target* are updated only if different than the corresponding element in *source*. For elements that point to allocated memory, **DtPrintCopySetupData** allocates new memory for those elements updated in *target*. Existing elements in *target* are freed using **XtFree**. All elements in a **DtPrintSetupData** structure can be freed by calling **DtPrintFreeSetupData**.

If *source* or *target* is **NULL** the copy will not be performed.

EXAMPLES

Sample code can be found in the `/proj/cde/examples/dtprint` directory.

RELATED INFORMATION

See the "DtPrintSetupBox" and "DtPrintFreeSetupData" sections in this chapter.

CDEnext

10.11 DtPrintFreeSetupData

10.11.1 Short Description

Free the memory pointed to by **`DtPrintSetupData`** structure elements.

10.11.2 Long Description

NAME

DtPrintFreeSetupData - free memory pointed to by **`DtPrintSetupData`** structure elements

SYNOPSIS

```
#include <Dt/Print.h>

void DtPrintFreeSetupData(
                            DtPrintSetupData *target)
```

ARGUMENTS

`target` points to the **`DtPrintSetupData`** structure whose elements are to be freed.

RETURN VALUE

None.

DESCRIPTION

`DtPrintFreeSetupData` calls **`XtFree`** to deallocate memory pointed to by elements of the **`DtPrintSetupData`** structure indicated by *`target`*. The **`DtPrintSetupData`** structure pointed to by *`target`* is not altered by this function.

EXAMPLES

Sample code can be found in the `/proj/cde/examples/dtprint` directory.

RELATED INFORMATION

See the “DtPrintSetupBox” section in this chapter for a definition of the **`DtPrintSetupData`** structure.

10.12 DtPrintResetConnection

10.12.1 Short Description

DtPrintResetConnection is a convenience function provided by the **DtPrintSetupBox** widget that allows applications to direct the widget to stop managing the print display connection. A mode parameter is included in order to direct the widget to close the print connection by calling **XpDestroyPrintContext** and **XtCloseDisplay** or to simply relinquish control of the connection without closing it.

10.12.2 Long Description

NAME

DtPrintResetConnection - reset the print display connection managed by a **DtPrintSetupBox**

SYNOPSIS

```
#include <Dt/Print.h>

XtEnum DtPrintResetConnection(
                              Widget wid,
                              DtPrintResetConnectionMode mode);
```

ARGUMENTS

wid — The **DtPrintSetupBox** widget ID.

mode — Indicates whether **DtPrintResetConnection** should close the X print server connection, or simply cause the **DtPrintSetupBox** to cease managing the connection.

RETURN VALUE

DtPRINT_SUCCESS
: **DtPrintResetConnection** was successful.

DtPRINT_NO_CONNECTION
: An open X print server connection is not currently being managed by the **DtPrintSetupBox**.

DtPRINT_BAD_PARM
: The value passed for *wid* is **NULL**, or an invalid *mode* was passed.

DESCRIPTION

DtPrintResetConnection is a convenience function provided by the **DtPrintSetupBox** widget that allows applications to direct the widget to stop managing the X print server connection. A mode parameter is included in order to direct the widget to close the print connection by calling **XpDestroyPrintContext** and **XtCloseDisplay** or to simply relinquish control of the connection without closing it.

CDEnext

DtPrintResetConnection is intended to be used by applications that fork a child process to perform the print rendering operation. Immediately after the fork is performed, the parent process will close its X print server connection, and retain its connection to the video X server. The forked child on the other hand will close its video X server connection and perform the rendering operation on the X print server connection.

Valid values for the *mode* parameter are:

DtPRINT_CLOSE_CONNECTION
: Set by the parent process when the application forks a child to perform the print rendering. This will cause the **DtNclosePrintDisplayCallback** list set for the passed **DtPrintSetupBox** to be called.

DtPRINT_RELEASE_CONNECTION
: Set when the application wishes to destroy the **DtPrintSetupBox** widget instance and still perform print rendering using the X print server connection initiated by the widget. For example, the child process of an application that forks to perform print rendering will close the video display connection (thereby destroying the **DtPrintSetupBox** widget) prior to print rendering. No **DtPrintSetupBox** callbacks will be called as a result of this operation.

EXAMPLES

Sample code can be found in the `/proj/cde/examples/dtprint` directory.

10.13 DtPrintSetupProc

10.13.1 Short Description

Type definition for procedure resources of the **DtPrintSetupBox** widget.

10.13.2 Long Description

NAME

DtPrintSetupProc - type definition for procedure resources of the **DtPrintSetupBox** widget

SYNOPSIS

```
typedef XtEnum (*DtPrintSetupProc)(
                              Widget wid,
                              DtPrintSetupData *print_data);
```

ARGUMENTS

wid The widget ID of the **DtPrintSetupBox**.

print_data A pointer to a **DtPrintSetupData** structure containing print setup information relevant to the specific procedure.

RETURN VALUE

DtPRINT_SUCCESS
The procedure completed successfully.

DtPRINT_FAILURE
The procedure encountered an error.

DtPRINT_BAD_PARM
An invalid parameter was passed to the procedure.

DESCRIPTION

DtPrintSetupProc is the type definition used for **DtPrintSetupBox** procedure resources. Each procedure is passed the widget ID of the **DtPrintSetupBox** via *wid*, and a structure containing information needed to perform the particular operation via *print_data*. If a procedure needs to update the **DtPrintSetupBox** it should do so by setting resources as indicated by the procedure resource description. The only exception to this is when the **DtNverifyPrinterProc** is used to verify X printers. In this case, the proc may update the *print_display* and *print_context* elements of the passed **DtPrintSetupData** structure.

EXAMPLES

Sample code can be found in the `/proj/cde/examples/dtprint` directory.

CDEnext

FILES

The **DtPrintSetupProc** type is defined in the Dt/Print.h include file.

RELATED INFORMATION

See the “DtPrintSetupBox” section in this chapter.

10.14 DtPrinterSelectionDialog

10.14.1 Short Description

The **`DtPrinterSelectionDialog`** is provided to allow the user to select an X printer from a complete list of printers for the client's video server, provided it supports the Xp extension, plus each server indicated by either the **XpServerList** resource or the XPSERVERLIST environment variable. The user typically selects one of the printers and chooses the OK button to return the selected printer to the caller.

10.14.2 Long Description

NAME

DtPrinterSelectionDialog - dialog for X Printer selection

DESCRIPTION

The **`DtPrinterSelectionDialog`** is provided to allow the user to select an X printer from a complete list of printers for the client's video server, provided it supports the Xp extension, plus each server indicated by either the **XpServerList** resource or the XPSERVERLIST environment variable. The user typically selects one of the printers and chooses the OK button to return the selected printer to the caller.

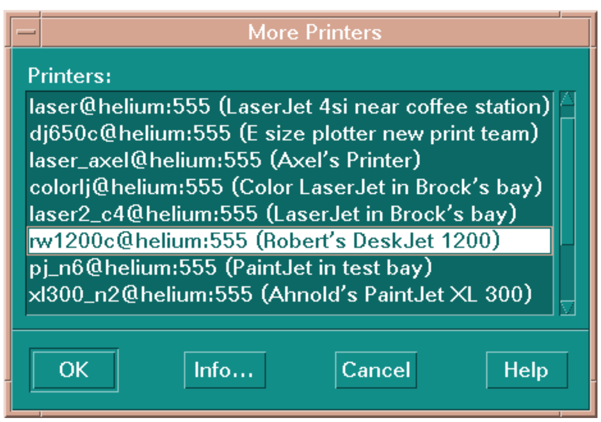


Figure 10-1. Printer Selection Dialog

The next section describes each of the **`DtPrinterSelectionDialog`** controls in detail.

Control Descriptions

Printers List

This is a list box containing all of the X Printers available on the print servers.

If the X server used to display this dialog is also an X print server, its printers will appear at the front of the list. Printers from additional servers specified by the **XpServerList** XRM resource or the XPSERVERLIST environment variable follow.

In order to select a printer to return to the caller, the user can single-click on the desired item, and activate the OK button, or simply double-click on the desired item.

OK Button

When the OK button is selected, the dialog is dismissed, and the currently selected X printer is returned to the caller.

Printer Information Button

Activating the “Info...” button displays the **`DtPrinterInfoDialog`**. This dialog presents additional information about the currently highlighted X Printer in the list. The information fields presented are the printer description, the printer model, the X Printer Specifier, and the document format used to generate documents sent to this X Printer.

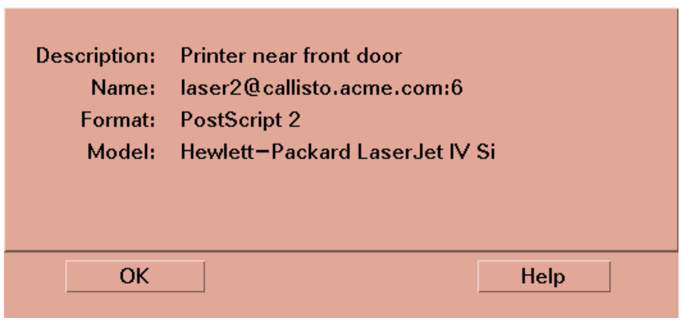


Figure 10-2. DtPrinterInfoDialog - X Printer Information

Cancel Button

When the Cancel button is selected, the dialog is dismissed, and no X printer name is returned.

EXTERNAL INFLUENCES

The list presented by the Printer Selection Dialog is influenced by either the **XpServerList** XRM application resource or the XPSERVERLIST environment variable. If specified, the resource takes precedence. See the “DESCRIPTION” section above for more information.

SEE ALSO

- The “DtPrintSetupBox” section in this chapter.

APPENDIX B Glossary

10.15 Fundamental DT Print Service Terms

X Print Extension
Structured as a standard X extension, the extension that provides functionality to delineate “print jobs” and “pages”.

X Print Extension API
The end-user API representing the X Print Extension.

X Print Extension Protocol
The wire-protocol existing between the X Print Extension API and the X Print Server.

DT Print Dialog Manager
A stand-alone component that interfaces with the X Print Service and provides setup dialogs used to configure a print job.

X Print Service
The all inclusive term that describes the collection of components which allow X rendering on non-display devices. The core components include: the X Print Server, the DT Print Dialog Manager, and the X Print Extension API.

X Print Server
A standard X server that has been modified to include the X Print Extension and any number of ddx drivers that can generate page description languages.

X Printer A printer served by an X Print Server.

X Printer Specification
A unique identifier for an X Printer. The format of the specifier is *printerName*`@`*host*`:`*display*.

10.16 Other non DT Print Service Specific Terms

ddx Short for device dependent X, a single module in an X-Server which is capable of driving a particular piece of display hardware, or in the case of the X Print Service, of generating a page description language (e.g. PCL).