X Print Service Extension Library

Protocol Version 1.0

X Consortium Standard
X Version 11 Release 6.4

A. Deininger
T. Gilg
J. Miller
H. Phinney
C. Prince

Hewlett-Packard Co.

K. Samborn
R. Swick

X Consortium, Inc.

Table of Contents

1 X Print Service Overview

The X Print Service allows X imaging to non-display devices, such as printers. It is called the X *"Print"* Service because the technology will primarily be applied to printing. The technology can, however, be applied 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 by extending the use of the X imaging model.

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 with small changes.

1.1 X Print Service Core Components

The X Print Service is made up of the following core components:

- 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.
- X Print "DDX" Drivers - DDX-level drivers for the X Server that generate page description languages (PDL) 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 components that are not included in this standard:

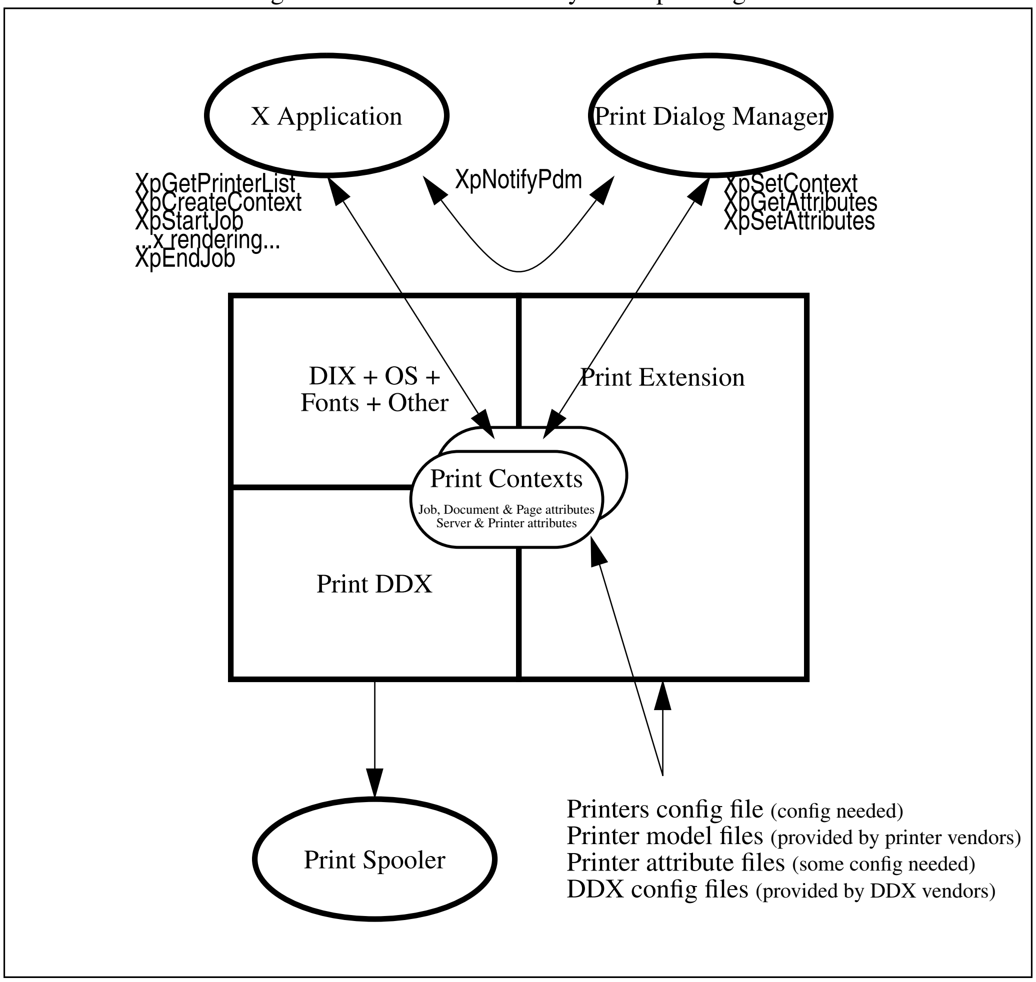
- libDtPrint - A library of print-specific GUIs tuned to several reference page-description-languages and printer models. See the Common Desktop Environment Specification, Version 2.
- dtpdm - Also known as the Dt Print Dialog Manager, a daemon-like process that provides secondary printer-specific GUIs that handle specific printer and spooler setup tasks. See the Common Desktop Environment Specification, Version 2.

Several keywords and concepts used in this specification were borrowed from the abstract standard ISO 10175, the subsetted standard and implementation represented by POSIX 1387.4, and the yet further subsetted implementation represented by OSF 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. It does, however, attempt to allow implementations to work with these print subsystem, and its architecture is open enough to allow tighter binding to a specific print subsystem in the future.

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 0-1.X Print Service Key Concepts Diagram



When the *X Print Server* starts, it may read a configuration file for instructions concerning 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 describe the capabilities of the printer models. Parallel to the printer model configuration files are some printer attribute configuration files which 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 by way of the XpGetPrinterList request. Once the application has selected a printer, it can create and set a *Print Context* using XpCreateContext and XpSetContext.

The *Print Context* represents the embodiment of the printer selected. It is initialized by the *X Print Server* at XpCreateContext time to contain a printer's default capabilities as well as the description of its overall capabilities, and to maintain the state of settings on the printer, the state of rendering against the printer, and 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 aspects of a Print Context are the *attribute pools* contained within it. These attributes express and control server, printer, job, document and page options. Attribute pools can be accessed and modified 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. The convenience routine XpGetPdmStartParams is provided to enlist the help of the *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 resulting PDL is either sent to a print spooler or can be retrieved by the application.

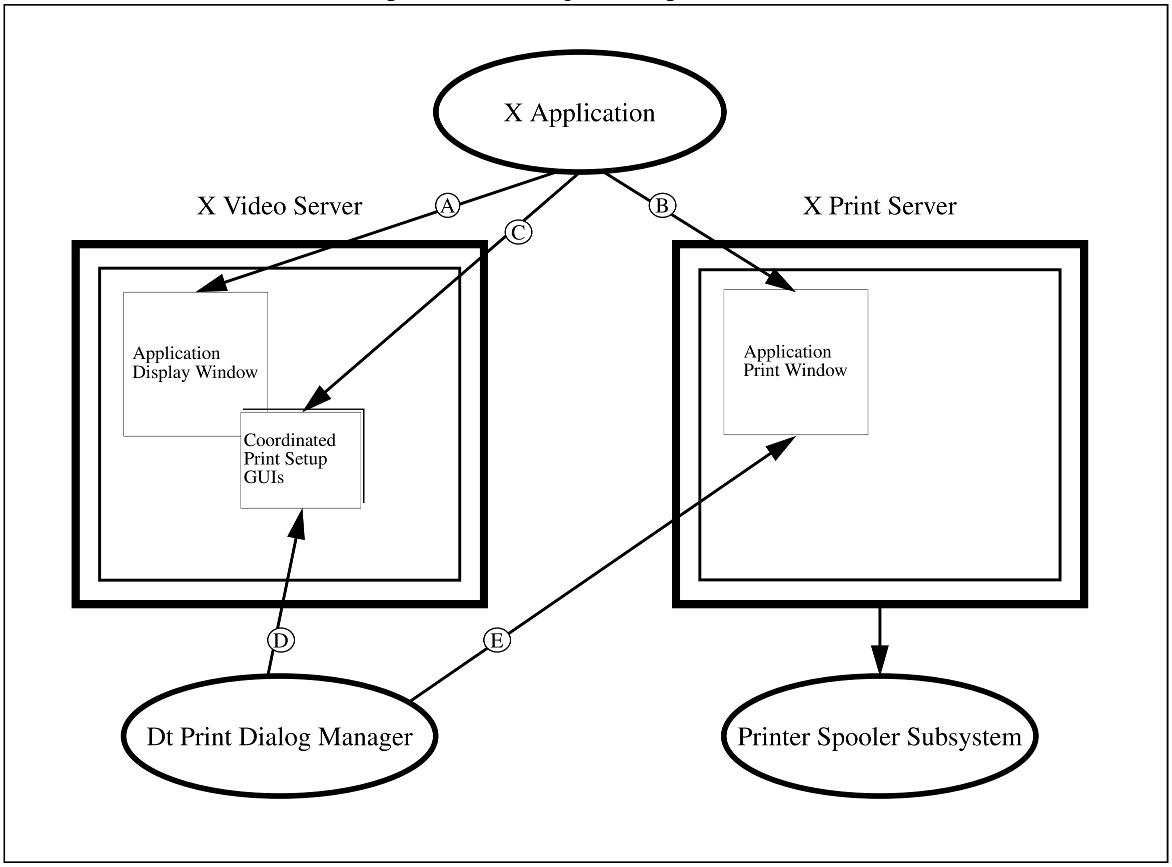
1.3 The Developer's/Integrator's View

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* in the following diagram). The primary difference is that when connected to the *X Print Serve*r additional calls can be made to delineate print "jobs", "documents" and "pages", and to create and modify a Print Context. The functions of the two servers may be combined into a single process, but applications will usually find it convenient to open separate connections for video and print rendering.

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 0-2. Developer's/Integrator's View



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

```
/*
 * 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 );
```

1.4 The Printer Vendor's View

The printer vendor is the person or company that wishes to enhance the X Print Service to support a new printer model or a new page description language. Enhancements may range from simple ones such as providing new printer model configuration files, to more complex ones such as providing a new *DDX driver* and corresponding *Print Dialog Manager*.

The major elements within the X Print Service that can be enhanced 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 an enhancement to an existing Print Dialog Manager. It can be used to provide dialogs that expose highly printer-specific options to the user and that communicate with the *DDX driver* by way of the Print Context attributes.
- The printer model files. These files describe the capabilities and defaults of printers based on the model.

1.5 The System Administrator's View

The system administrator is the person who configures and maintains the system processes and files associated with the X Print Service. An X Print Service implementation will typically have built-in fallback defaults for nearly everything, but in custom environments it will be configured considerably.

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 *Print Dialog Manager*. Using this architecture support for new page description languages and spooler subsystems can be added centrally, without reconfiguring applications.

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

The key areas of configuration and system administration are:

- X Print Service Startup - Deciding whether a "per-user " or "global service" model of operation is desired. In the per-user model, a separate X Print Server process with its own Print Dialog Manager exists for each desktop. In the global service model, a centralized X Print server process services multiple users in a workgroup. Typically, there may be one such centralized process per shared printer.
- *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 duplex printing option because the printer's duplexer isn't working).

2 X Print Service Extension Library

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 Protocol are provided which make it easier for an application programmer to use the X Print Service.

The X Print Service Extension Library concentrates on print job, document and page management. It includes the following calls:

- 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
- XpSetLocaleHinter
- XpGetLocaleHinter

2.1 Dependencies

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

2.2 Library Calls

The header file X11/extensions/Print.h contains prototypes for the following routines.

2.2.1 Creating and Managing Print Contexts

Use XpCreateContext to create and initialize a new print context.

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

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.

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 is 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 in it can be accessed and modified by calling XpGetAttributes and XpSetAttributes, and the event selections in it 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.

The XPContext returned by XpCreateContext is an XID, and can be used to set the print context on display connections by calling XpSetContext. The XPContext id can be shared between processes and display connections. It is the responsibility of the clients sharing a print context to coordinate their usage of the context; for example they must ensure 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, the application must query the *X Print Server* via XpGetScreenOfContext for the screen that has been associated with the print context, and then create all server resources that will be used in the print job on that screen. Failure to do so will result in undefined behavior.

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 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* chooses a default value.

This function can generate a BadMatch error if the specified printer_name does not exist on display, or if the print server could not interpret the code set specified in printer_name.

Use XpSetContext to set or unset a print context with the specified display connection to the *X Print Server.*

void XpSetContext (*display*, *print_context*)
    Display **display*;

XPContext *print_context*;

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

print_context A pre-existing print context on the same X Server.

XpSetContext sets the print context for 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 multiple jobs, if not destroyed.

If print_context is None, XpSetContext 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, XpSetContext may modify the list of available fonts (see XListFonts) on display, and the actual set of usable fonts (see XLoadFont). A unique combination of fonts may be available from within a given print context; a client should not assume that all the fonts available when no print context is set will be available when a print context is set.

When a print context is set on a display connection, the default behavior of ListFonts and ListFontsWithInfo 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. The xp-listfonts-modes attribute is provided so that applications can control the behavior of ListFonts and ListFontsWithInfo and is typically used 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.

If the value of xp-listfonts-modes includes xp-list-glyph-fonts, ListFonts and ListFontsWithInfo will include all of the fonts available to the server that have glyphs associated with them. If the value of xp-listfonts-modes includes xp-list-internal-printer-fonts, then ListFonts and ListFontsWithInfo will include all of the fonts defined as internal printer fonts.

When the print context is unset or XpDestroyContext is called, the available fonts on display revert back to what they were previously.

XpSetContext can generate an XPBadContext error.

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

XPContext XpGetContext (*display*)
Display **display*;

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

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.

Use XpDestroyContext to unset and destroy a print context.

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

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

print_context Specifies the print context to destroy.

XpDestroyContext closes any outstanding associations between the print context and any display connections, and then destroys the print context. All display connections using the print context will no longer be able to access the print context.

Destroying a print context will cause any in-progress pages, documents and jobs to be canceled within the *X Print Server.*

XpDestroyContext can generate an XPBadContext error.

2.2.2 Obtaining the Screen for a Print Context

Use XpGetScreenOfContext to obtain a pointer to the screen associated with the specified print context.

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

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
print_context	A pre-existing print context. This argument is currently ignored, but must be specified.

XpGetScreenOfContext returns the screen that is associated with the current print context of display. This call must be made after XpSetContext to determine which specific screen other X resources must be created on.

Each printer supported by a print server is associated with exactly one of the screens returned in the connection setup reply.

XpGetScreenOfContext will generate an XPBadContext error if print_context is invalid.

2.2.3 Obtaining Page Dimensions

Use XpGetPageDimensions to get the page dimensions for the current printer settings.

Status XpGetPageDimensions (*display*, *print_context, width, height, reproducible_area*)
    Display **display*;
    XPContext *print_context*;
    unsigned short **width*;
    unsigned short **height*;
    XRectangle**reproducible_area*;

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

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. The net reproducible area is the portion of the page on which the printer is physically capable of placing ink.

XpGetPageDimensions returns a Status of 0 on failure, or 1 on success.

XpGetPageDimensions can generate an XPBadContext error.

Use XpSetImageResolution to set the resolution for subsequent PutImage requests.

Bool XpSetImageResolution (*display*, *print_context, image_res, prev_res_return*)
  Display **display*;
  XPContext *print_context*;
  int *image_res*;
  int **prev_res_return*;

display	Specifies a pointer to the Display structure.
print_context	Specifies the print context on which to set the resolution.
image_res	Specifies the image resolution in pixels per inch.
prev_res_return	Returns the previous image resolution in pixels per inch.

XpSetImageResolution returns True if the printer server allowed the resolution to be set, otherwise False is returned.

XpSetImageResolution sets the resolution for subsequent PutImage requests to the screen of the specified print context. If the return value is False, then the print server does not support image scaling for the particular resolution given the current configuration of the printer, and the application is responsible for any desired scaling. If the return value is True, then the contents of any subsequent PutImage request to a Pixmap or to a Window on the screen of the specified print context will automatically be scaled as part of the PutImage request. The scale factor is:

  default_printer_resolution / image_res

Where default_printer_resolution is the current value of that page attribute. Only the image itself is scaled (meaning the effective width and height of the image change), the dst-x and dst-y parameters to PutImage are not altered.

As a special case, a value of zero for image_res resets the resolution to automatically track the printer resolution; in this case (which is also the default setting for a newly created print context), subsequent images will not be scaled.

If the return value is True and prev_res_return is a non-NULL pointer, then the previous image resolution that was set for the print context is stored in prev_res_return.

XpSetImageResolution returns False immediately if image_res is negative or greater than 65535.

XpSetImageResolution can generate an XPBadContext error.

Use XpGetImageResolution to get the current image resolution for a print context.

int XpGetImageResolution (*display*, *print_context*)
  Display **display*;
  XPContext *print_context*;

display	Specifies a pointer to the Display structure.
print_context	Specifies the print context on which to get the resolution.

XpGetImageResolution returns the current image resolution for the specified print context. A value of zero means the resolution automatically tracks the printer resolution. If the request fails in some way, a negative value is returned.

XpGetImageResolution can generate an XpBadContext error.

2.2.4 Starting, Ending, and Canceling Jobs

Use XpStartJob to indicate the beginning of a single print job.

void XpStartJob *(display, output_mode)*
    Display **display*;
    XPSaveData *output_mode*;

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
output_mode	Specifies how the printer output data is to be handled.

XpStartJob signals the beginning of a new print job.

If output_mode is XPSpool the *X Print Server* will automatically spool the printer output. If output_mode is XPGetData, then the *X Print Server* buffers the document output for retrieval by XpGetDocumentData. In this case, the print server suspends processing further requests on this print context until some other client sends XpGetDocumentData**.** Subsequent operations that use the print context may be suspended at any time pending the processing of XpGetDocumentData replies to read any buffered output.

The XPSaveData values for output_mode are defined in <X11/extensions/Print.h>.

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

XpStartJob sets the job-owner job attribute (included in the XPJobAttr pool) immediately prior to issuing the PrintStartJob request. On POSIX systems, the job-owner attribute is set using getpwuid_r on the result of getuid. This attribute may be used by the *X Print Server* to identify the user to the spooler.

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

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPStartJobNotify when the *X Print Server* has completed the PrintStartJob request.

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*, these delineations may be mapped by a DDX driver into real functionality (e.g. see the server attribute multiple-documents-supported).

XpStartJob 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 (for example, XpEndJob prior to XpStartJob).
BadValue	The value specified for output_mode is not valid.

Use XpEndJob to indicate the ending of a single print job.

void XpEndJob *(display)*
    Display **display*;

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

XpEndJob signals the end of a print job. Any accumulated print data that remains is either sent to the printer or made available to XpGetDocumentData.

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

XPEndJobNotify indicates that the document data has been sent to the spooler (output_mode=XPSpool) or been completely sent to the client via XpGetDocumentData (output_mode=XPGetData) - it does not mean that the document data has been completely received and processed by the client or spooler.

XpEndJob 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 (for example, XpEndJob prior to XpStartJob).

Use XpCancelJob to cancel a single print job.

```
void XpCancelJob (display, discard)
    Display *display;
    Bool discard;
```

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
discard	When TRUE, specifies that all XPPrintNotify events should be discarded.

XpCancelJob cancels an in-progress job. If the job was started with output_mode XPGetData then the data stream to XpGetDocumentData is terminated. For many page description languages such arbitrary termination may invalidate the output.

If the job was started with output_mode XPSpool then depending on the driver and spooler configuration the entire job may be canceled or a partial job may be generated.

If discard is TRUE, all XPPrintNotify events with a detail field of XPEndPageNotify, XPEndDocNotify, or XPEndJobNotify are discarded before XpCancelJob returns.

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPEndJobNotify.

XpCancelJob 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 (for example, XpEndJob prior to XpStartJob).

2.2.5 Starting, Ending, and Canceling Documents

Use XpStartDoc to indicate the beginning of a print document.

```
void XpStartDoc (display, type)
    Display *display;
    XPDocumentType type;
```

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
type	Specifies the type of document. It can be either XPDocRaw or XPDocNormal.

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 an 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 are defined in <X11/extensions/Print.h>:

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

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

The 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 optional. If XpStartPage is called immediately after XpStartJob then a synthetic XpStartDoc with XPDocNormal will be assumed by the *X Print Server* prior to XpStartPage (i.e. the XPStartDocNotify and XPStartPageNotify events will have the same sequence number). Likewise, if XpEndJob is called immediately after XpEndPage then a synthetic XpEndDoc will be assumed by the *X Print Server* prior to XpEndJob (i.e., the XPEndDocNotify and XPEndJobNotify events will have the same sequence number).

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPStartDocNotify.

XpStartDoc 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, XpStartDoc prior to XpStartJob).
BadValue	The value specified for type is not valid.

Use XpEndDoc to indicate the ending of a print document.

void XpEndDoc *(display)*
    Display *display;

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

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

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPEndDocNotify.

XpEndDoc 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).

Use XpCancelDoc to cancel a print document.

void XpCancelDoc *(display, discard)*
    Display **display*;
    Bool *discard*;

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
discard	When TRUE, specifies that all XPPrintNotify events with a detail of XPEndPageNotify or XPEndDocNotify should be discarded.

XpCancelDoc cancels an in-progress document. If the job was started with output_mode XPGetData then the data stream to XpGetDocumentData is interrupted; no further data for the current document will be generated but data for subsequent documents can be generated. For many page description languages such arbitrary termination may invalidate the output.

If the job was started with output_mode XPSpool then depending on the driver and spooler implementation the entire document may be canceled or a partial document may be generated.

If discard is True all XPPrintNotify events with a detail field of XPEndPageNotify or XPEndDocNotify are discarded before XpCancelDoc returns.

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPEndDocNotify.

XpCancelDoc 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).

2.2.6 Getting and Putting Data into Documents

Use XpPutDocumentData to send and incorporate data into the output.

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

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

Depending on type for XpStartDoc, XpPutDocumentData has two modes of operation.

In XPDocRaw mode, XpPutDocumentData sends data directly to the output, and drawable must be None, else a BadDrawable error 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 using the *X Print Server*'s job control and submission capabilities. The printer attribute xp-raw-formats-supported defines the valid values for doc_fmt in this mode, with unsupported values for doc_fmt causing a BadMatch error 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 a BadMatch error to be generated.

If doc_fmt is not in either xp-raw-formats-supported or xp-embedded-formats-supported a BadValue error is generated.

Depending on the DDX driver implementation in use, XpPutDocumentData might 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 might 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 value of data_len. XpPutDocumentData automatically decomposes the call into multiple protocol requests to make sure that the maximum request size of the server is not exceeded.

XpPutDocumentData 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 (for example, XpPutDocumentData prior to XpStartDoc).
BadValue	The value specified for doc_fmt is not supported.
BadMatch	The value specified for doc_fmt is not valid for the current document type or the value specified for drawable is not valid for the print context and print screen.
BadDrawable	The value specified for drawable is not valid.

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

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

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 procedure to be registered and called repeatedly to save blocks of document data.
finish_proc	A procedure to be registered and called once when the print job has completed and all document data has been sent to save_proc.
client_data	Specifies client data to be passed to save_proc and finish_proc when called.

The return value is NULL if XpGetDocumentData encounters an error, non-NULL otherwise.

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 by way of *different* display connections. Though XpGetDocumentData retrieves document data, its effect is bounded by XpStartJob and XpEndJob. XpGetDocumentData always returns immediately; if an error occurs and the callbacks cannot be registered, the return status is 0, else the return status is non-zero and the callbacks will be called sometime after the return from XpGetDocumentData. This producer/consumer exchange is set up when XpStartJob is called by the producer with output_mode equal XPGetData, and is subsequently initiated when XpGetDocumentData is called by the consumer. Though XpStartJob will return immediately, further attempts to use the producer's display connection may be blocked by the *X Print Server* until XpGetDocumentData is called on the consumer's display connection.

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.

The save_proc is defined in <X11/extensions/Print.h> as:

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

The save_proc is repeatedly called on each chunk of document data sent by the *X Print Server* until either XpEndJob or XpCancelJob is called. data_len specifies the number of bytes in data. The memory for data itself is owned by the library, 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 finish_proc is defined in <X11/extensions/Print.h> as:

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

After XpGetDocumentData successfully registers the callbacks, any generated X errors (for example, BadAlloc) or Xp errors (for example, XPBadContext or XPBadSequence) that are the result of XpGetDocumentData will cause the Xlib error handler to be invoked, and then 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 an XPBadSequence error is generated and finish_proc is called with XPGetDocError. If XpGetDocumentData is called after XpStartJob and output_mode was specified as XPSpool, then an XPBadSequence error is generated and finish_proc is called with XPGetDocError. If the producer starts generating data and the consumer cannot consume data quickly enough, then the producer's display connection will be blocked by the *X Print Server.*

Until XpEndJob or XpCancelJob is called, it is possible that various XPPrintNotify events will be generated (for example, a page has been canceled). The data passed to save_proc is not necessarily organized according to the consumer's requests or any generated events, and its consistency is guaranteed only if the entire job completes successfully (i.e. without being canceled or generating an error). Consumers may want to select for XPPrintNotify events and terminate save processing upon receipt of cancellation events.

When finish_proc is called, sometime after XpGetDocumentData is called and returns, status gives the completion status of the job and is defined in <X11/extensions/Print.h> as:

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

XPGetDocFinshed 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, save_proc and finish_proc are unregistered and will no longer be called.

XpGetDocumentData can generate one of the following errors:

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 (for example, XpGetDocumentData prior to XpStartJob).

2.2.7 Starting, Ending, and Canceling Pages

Use XpStartPage to indicate the beginning of a print page.

void XpStartPage *(display, window)*
    Display **display*;
    Window *window*;

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
window	Specifies the window ID.

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

XpStartPage causes window to be mapped. See XpGetPageDimensions and XResizeWindow for details on resizing window to the size of the media selected prior to calling XpStartPage. Within the XpStartPage and XpEndPage sequence, attempts to resize, move or unmap window will yield undefined results. To resize or move inferiors of window the standard semantics of ConfigureWindow apply, except that the contents of any configured window may be discarded. An Expose event will be generated if a window's contents are discarded.

All changes to the XPPageAttr attribute pool (see XpSetAttributes) must be made prior to calling XpStartPage, after which an XPBadSequence error 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 when the *X Print Server* has completed XpStartPage. If the event Expose is also selected for (see XSelectInput), the exposure events will be generated prior to XPPrintNotify.

The client need not wait for XPStartPageNotify prior to calling any other X rendering routines.

XpStartPage 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; for example, XpStartPage was called before XpStartJob or was called for a type XPDocRaw document.
BadWindow	The value specified for window is not valid.

Use XpEndPage to indicate the end of a print page.

void XpEndPage *(display)*
    Display **display*;

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

XpEndPage signals the end of a print page, and causes window to be unmapped. All resulting page data is assembled and combined with data previously sent by XpPutDocumentData. No generation of document data will occur for rendering operations to the corresponding windows after XpEndPage is called.

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPEndPageNotify when the *X Print Server* has completed XpEndPage.

XpEndPage can generate an XPBadSequence error.

Use XpCancelPage to cancel a print page.

void XpCancelPage *(display, discard)*
    Display **display*;
    Bool *discard*;

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

discard When TRUE, specifies that XPPrintNotify events with a detail of XPEndPageNotify should be discarded.

XpCancelPage cancels an in-progress page. If the job was started with output_mode XPGetData then the data stream to XpGetDocumentData is interrupted; no further data for the current page will be generated but data for subsequent pages can be generated. For many page description languages, such arbitrary interruptions may invalidate the output.

If the job was started with output_mode XPSpool then depending on the driver and spooler implementation the entire page may be canceled or a partial page may be generated.

If discard is True all XPPrintNotify events with a detail field of XPEndPageNotify are discarded before XpCancelPage returns.

For clients selecting XPPrintMask (see XpSelectInput), the event XPPrintNotify will be generated with its detail field set to XPEndPageNotify when the *X Print Server* has completed XpCancelPage.

XpCancelPage can generate an XPBadSequence error.

2.2.8 Selecting Input

Use XpSelectInput to select which X Print events from the specified print context the client is interested in.

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

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 one or more of the valid events mask bits (see below).

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
```

XpSelectInput can generate one of the following errors:

XPBadContext The specified print context is not valid.

BadValue The value specified for event_mask is not valid.

Use XpInputSelected to query which X Print events the client has selected to receive from the specific print context.

unsigned long XpInputSelected *(display, context, all_event_mask_return)*
 Display **display*;
 XPContext *context*;
 unsigned long **all_event_mask_return*;

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
context	Specifies the print context to which the query is being made.
all_event_mask_return	Returns the set of events any client has selected.

This request returns a bit mask describing which event classes the client has selected to receive. The value returned to all_event_mask_return is the union of every client's event mask.

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.

XpInputSelected can generate an XPBadContext error.

2.2.9 Getting and Setting Attributes

Use XpGetAttributes to get an attribute pool from the specified print context.

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

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

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.

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 File Syntax" in the Xlib specification).

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.

XpGetAttributes 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.

If any errors occur, XpGetAttributes returns NULL.

Use XpGetOneAttribute to get a single print attribute from the specified print context.

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

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

This request returns a COMPOUND_TEXT string attribute_value, else NULL if any errors occurred.

XpGetOneAttribute is a variation of XpGetAttributes to get a single attribute value from an attribute pool. Unlike XpGetAttributes, where the reply contains an entire attribute pool, XpGetOneAttribute returns just one attribute_value.

attribute_name should not include a colon. The caller is expected to free the attribute value returned using XFree.

XpGetOneAttribute 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.

Use XpSetAttributes to set or update an attribute pool in the specified print context.

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

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
context	The print context whose attribute pool is to be modified.
type	Specifies the attribute pool to be modified.
pool	An attribute pool represented as a resource string. Encoded in COMPOUND_TEXT.
replacement_rule	Either XPAttrReplace orXPAttrMerge.

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 existing attribute pool is discarded and 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 afterwards.

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
```

When setting supported attribute names, the X Print Server and associated driver will validate the new values and ignore those that are invalid; previous values remain unchanged. 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 explicitly or implicitly resets the attribute to a default value.

When setting certain supported attributes, the X Print Server may 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. For attribute pools that are writable, lists of the supported attributes can be found in the XPPrinterAttr pool.

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 be retained 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.

Refer to 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.

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 operations.

XpSetAttributes 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 occurred 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.

2.2.10 Getting Printer Lists

Use XpGetPrinterList to retrieve a list of all printers supported on an *X Print Server.*

```
XPPrinterList XpGetPrinterList (display, printer_name, list_count_return)
      Display *display;
      char *printer_name;
      int *list_count_return;
```

display Specifies a pointer to the 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_return Returns the number of printers in the list.

XpGetPrinterList returns a list of printer records where each record describes a printer supported by the *X Print Server,* or NULL if any errors occur.

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 is 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 caller'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 for each printer in the list. If the *X Print Server* cannot understand the hint, the *X Print Server* will choose a default.

The returned printer list can be freed by calling XpFreePrinterList.

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

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

XpGetPrinterList can generate a BadAlloc error.

XpFreePrinterList should be used to free a printer list.

```
void XpFreePrinterList (printer_list)
      XPPrinterList printer_list;
```

printer_list A list of printer records returned by XpGetPrinterList.

XpFreePrinterList frees the list of printer records returned by XpGetPrinterList.

Use XpRehashPrinterList to recompute the list of available printers.

```
void XpRehashPrinterList (display)
```

Display **display*;

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

XpRehashPrinterList causes the *X Print 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 Print Server*, the *X Print 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.

Existing print contexts will not be affected by XpRehashPrinterList as long as their printer destination remains valid.

2.2.11 Querying Version, Extension, and Screen

Use XpQueryVersion to 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.

Status XpQueryVersion *(display, major_version_return, minor_version_return)*
Display **display*;
short **major_version_return*;
short **minor_version_return*;

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
major_version_return	Returns the major version if the X Print Service Extension exists, else zero.
minor_version_return	Returns the minor version if the X Print Service Extension exists, else zero.

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. If the extension is supported, the major and minor version numbers are returned to indicate the level of X Print Service Extension support.

The X Print Service Extension is initialized on the first call to any X Print Service function; there is no need to explicitly initialize the X Print Service Extension.

Use XpQueryExtension to 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.

Bool XpQueryExtension *(display, event_base_return, error_base_return)*
Display **display*;
int **event_base_return*;
int **error_base_return*;

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.

XpQueryExtension determines if the X Print Service Extension is present. It returns True if the extension is supported, otherwise False. 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 initialized on the first call to any X Print Service function; there is no need to explicitly initialize the X Print Service Extension.

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

Screen **XpQueryScreens *(display, list_count_return)*
    Display **display*;
    int **list_count_return*;

display	Specifies a pointer to the Display structure; returned from XOpenDisplay.
list_count_return	Returns the number of screens in the list.

This request returns a non-NULL pointer to a list of screen pointers if one or more screens support the X Print Service Extension; otherwise it returns NULL.

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 screen pointers can be freed by calling XFree.

2.2.12 Getting PDM Parameters

Use XpGetPdmStartParams as a standard convenience function to build up parameters in accordance with the PDM Selection Protocol

Status XpGetPdmStartParams *(print_display, print_window, print_context, video_display, video_window, selection_display_return, selection_return, type_return, format_return, data_return, nelements_return)*
    Display **print_display*;
    Window *print_window*;
    XPContext *print_context*;
    Display **video_display*;
    Window *video_window*;
    Display ***selection_display_return*;
    Atom **selection_return*;
    Atom **type_return*;
    int **format_return*;
    unsigned char ***data_return*;
    int **nelements_return*;

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 near which the transient dialogs from the PDM should be posted.
selection_display_return	Returns the display connection on 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_return	Returns the selection atom for which a PDM selection should be made.
type_return	Returns the type for the PDM Selection Protocol property the caller is expected to create.

format_return Returns the format for the PDM Selection Protocol property the caller is expected to create.

data_return Returns the data set for the PDM Selection Protocol property the caller is expected to create. The caller is expected to XFree the data when finished.

nelements_return Returns the number of elements for the PDM Selection Protocol property the caller is expected to create.

This request returns a zero status if an error occurred, non-zero otherwise.

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 is responsible for 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.

XpGetPdmStartParams returns zero if an error occurred, else non-zero. If an error occurs all other _return values are undefined.

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_return is set equal to print_display. If XPDMDISPLAY is set to one of the keywords "print" or "video", selection_display_return is set to print_display or video_display, respectively. If XPDMDISPLAY is set to a valid DISPLAY-style string, selection_display_return may be set, as appropriate, to one of print_display, video_display, or to a *new* display connection opened from within XpGetPdmStartParams. Only in the single case where a *new* display connection is made should the caller close selection_display_return using XCloseDisplay.

When XpGetPdmStartParams is called, the caller's locale (see XpSetLocaleHinter) is included in the 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 choose a default. Note that the locale of the print attributes that the PDM will subsequently access, will already have been determined when the client called XpCreateContext.

The environment variables XPDMDISPLAY and XPDMSELECTION are re-read each time XpGetPdmStartParams is called.

2.2.13 Setting and Getting Locale Hinters

Use XpSetLocaleHinter to set a "locale hinter" function and description of it.

void XpSetLocaleHinter *(hinter_proc, hinter_desc)*
    XPHinterProc *hinter_proc*;
    char **hinter_desc*;

hinter_proc A pointer to a "hinter proc".

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

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 to register more advanced hint generators. By default, Xp uses a hinter proc that calls setlocale 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 by the Xp calls themselves.

When the client’s locale is needed, 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 ‘;’.

Use XpGetLocaleHinter to get a pointer to and description of the current “locale hinter” function.

char *XpGetLocaleHinter *(hinter_proc_return)*
    XPHinterProc **hinter_proc_return*;

hinter_proc_return Returns a pointer to the current hinter proc.

XpGetLocaleHinter returns the currently installed hinter proc and hinter description. The function value is the pointer to the description. The caller is expected to XFree the returned hinter description string.