X Window System, Version 11

Xt Internationalization

Public Review Draft, March 18, 1991

(Send comments to i18n@expo.lcs.mit.edu)

Bill McMahon

Hewlett Packard Company

ABSTRACT

The following specification describes changes and additions to the Intrinsics for X11R5 to support development of internationalized clients.

The specifications are consistent with changes proposed to Xlib for X11R5 to support internationalization and with related specifications from X/Open Portability Guide, Release 3, and ANSI-C.

The reader is assumed to be familiar with those, particularly with the notion of locale in the C language, therefore they will not be detailed here.

This documentation is provided ‘‘as is’’ without express or implied warranty. This document is only a draft standard of the MIT X Consortium and is therefore subject to change.

1. Introduction

This proposal is broken into two sections. The first section describes the issues considered, the changes (if any) proposed, and the some of the reasoning behind the changes (or the lack of changes). The second section describes the changes to the Xt specs themselves.

1.1. Internationalization of Strings

Most vendors that have added internationalization to the R4 release of the Intrinsics have simply assumed that the type String referred to characters in the encoding of the locale. In order to have Xt continue to work in most current implementations without breaking existing applications, it was decided to formalize this assumption. Thus, previously existing references to String in the Intrinsics were not changed, but the meaning of the references has changed.

1.2. Establishing Locale

Internationalization as defined by ANSI is a technology that allows support of an application in a single locale. In adding support for internationalization to the X Window System the restrictions of this model have been followed. For this reason, no Xt routine makes a call to establish the locale.

It is considered possible to create a toolkit that could establish multiple locales, but this was not a design consideration for this specification.

A relevant fact from the Xlib I18N spec is that objects in X inherit the global locale at create time. An implication of this fact is that locale must be set before an object which needs that locale is created.

We need to consider the actions taken by XtDisplayInitialize in R4 to create the resource database for a display. First, XtDisplayInitialize creates two temporary databases (one from the command line, one from the default resource file). It then queries these databases for the value of the resource XnlLanguage in order to determine the language string. If the database queries fail, the language string is determined from the environment (for POSIX this is the LANG environment variable). It then uses this value to find the app-defaults file and creates the display resource database (merging in the temporary databases).

It is stated in the Xlib I18N spec (section 3.1.1) that a command line argument or a resource can be used to set the locale. The implication of this for Xt is that the client needs access to the language string so it may set the locale prior to Xt loading the app-defaults file.

It should be noted that the temporary database itself is associated with the active locale at the time it is built. Specifically, the "-xrm" option is parsed assuming whatever locale is set prior to loading the temporary database.

In order to provide access to the language resource, the Xt-I18N spec proposes to create a language procedure which is called by XtDisplayInitialize after the language resource has been determined and before the full resource database is created. There will also be a function to set this language procedure.

In addition, determining the language by directly examining an environment variable as was done in R4, is not the correct action for an internationalized Xt. The language should be determined from the operating systems locale mechanism. For POSIX, this is the setlocale function.

In order to use this interface, the locale should be established first. In addition, the application may wish to establish locale after the language resource has been determined.

It is also the responsibility of the client to call XSupportsLocale to ensure that the locale is supported. XSetLocaleModifiers should be called when a client needs to establish any local modifiers for X.

Here are two examples of how to establish locale using the ANSI-C setlocale mechanism. An application that wants to inherit the locale from the environment may use the following:

```
setlocale(LC_ALL,"");
if (!XSupportsLocale())
  fprintf(stderr,"Warning: initial process locale not supported");
top = XtAppInitialize( ... );
```

An application that wants to set its locale according to the language resource may use the following:

```
char * myLanguageProc(display, xnl, client_data)
Display * display;
char *  xnl;
caddr_t client_data;
{
  char *locale = setlocale(LC_ALL, xnl);
  if (!XSupportsLocale())
    fprintf(stderr, "Warning: locale %s not supported", locale);
  return locale;

}

setlocale(LC_ALL,"");
if (!XSupportsLocale())
  fprintf(stderr,"Warning: initial process locale not supported");
XtSetLanguageProc( NULL , myLanguageProc, NULL);
top = XtAppInitialize( ... );
```

1.3. Event Management

The only change to Xt's event management for internationalization is the invocation of filters that can be used by back-end input methods. Filters are needed to support back-end input methods in order to get events before they are dispatched by the Intrinsics. The need for filters is discussed in the "X Window System, Version 11 Input Method Specifications". These events will be filtered in **XtDispatchEvent**.

If there is a server grab for an event and XFilterEvent returns true for that event then the grab must be ungrabbed. The window passed to XFilterEvent is the window associated with the widget to which the event would be dispatched if there is no filter.

1.4. Resource Management

FontSets will need to be supported within Xt. There will need to be a String-To-FontSet converter. The converter will treat any return of a fontset as a valid return, but it will print a warning if there were any missing charsets.

There will need to be a resource name, class type, and representation type for font sets.

1.5. Translation Management

This specification proposes that the translation manager not be changed for R5. The changes proposed to support input methods in the XIM proposal are intended to support textual input. This is not the proper concern of the translation manager. Current translation syntax for supporting key sequences which are described as sequences of Latin-1 characters will be maintained without change.

1.6. Utility Functions

The Xt error database needs to be able to be localized. Unfortunately, the app-context interface to the error database does not contain a display. This makes it difficult to use **XtResolvePathname** to obtain the error database.

For this reason, the Xt-i18n specification will take the same approach as the Xlib i18n. It will be specified that the /usr/lib/X11/XtErrorDB is not necessarily used. That seems to be the strongest statement that can be made given the app-context approach to the error databases.

Since the language string returned by the operating system's locale mechanism may not be in the format specified for XtResolvePathname in R4, that syntax is no longer required. Thus, the precise syntax of the language string and the manner in which a "language part", "territory part" and "codeset part" of the language string are determined is now implementation dependent.

2. Specification Changes

This chapter describes the changes to the "X Toolkit Intrinsics - C Language Interface".

2.1. Widget Instantation

2.1.1. Initializing the Intrinsics and Establishing Locale.

The Xt-i18n spec proposes the addition of the following procedure type and the subsequent procedure that sets a value of this type.

No Intrinsics function shall call the operating system's locale mechanism to establish the locale. The locale should be established before any Intrinsics function is called. In addition, the Intrinsics allows an application to set the locale to the value of the language resource using a language procedure of type **XtLangProc**:

typedef String (*XtLangProc) (Display *, char *, caddr_t);
    Display **display*;
    char **xnl*;
    caddr *client_data*;

display	Specifies the display.
xnl	Specifies the language value.
client_data	additional client data passed to the language proc.

This procedure allows an application to set the locale to the value of the language resource determined by **XtDisplayInitialize.** The function returns a language string which will be subsequently used by **XtDisplayInitialize** to establish the path for loading resource files.

To set the language procedure for use by **XtDisplayInitialize** use **XtSetLanguageProc**.

XtLangProc XtSetLanguageProc(*app_context*, *proc*, *client_data*)
    XtAppContext *app_context*;
    XtLangProc *proc*;
    caddr *client_data*;

app_context	Specifies the application context.
proc	Specifies the language proc.
client_data	additional client data to be passed to the language proc.

The **XtSetLanguageProc** function sets the language procedure which will be called from **XtDisplayInitialize**. If app_context is NULL, it registers the specified language proc in all application contexts created by the calling process, including any future application contexts that may be created. It will return the previous registered language proc, or the default one if no language proc has been registered.

The default language proc is **XtDefaultLanguageProc** and it will return the language string if the language string is not NULL else it will return the language string determined by the operating system's locale mechanism; for POSIX-based systems, this is the return value from a call to setlocale(LC_ALL,NULL).

The Xt-i18n proposal proposes that **XtDisplayInitialize** be changed as follows to call the Intrinsics language procedure.

XtDisplayInitialize function retrieves the language string to be used for the specified display (see Section 11.11), /+ calls the language procedure with that language string, +/ builds the resource database, calls the Xlib **XrmParseCommand** function to parse the command line, and performs other per display initialization.

2.1.2. Loading the Resource Database

The Xt-i18n proposal proposes that this section be changed as follows to specify that the default locale string be determined from the language procedure.

... The database constructed from the command line is then queried for the resource *name*.**xnlLanguage**, class *Class*.**XnlLanguage** where name and Class are the specified application_name and application_class. If this database query fails, the server resource database is queried;

/- if this query also fails, the language is determined from the environment; on POSIX-based systems this is done by retrieving the value of the **LANG** environment variable. If no language string is found, the empty string is used. -/

/+ if this query also fails, the language string is initially set to NULL. The Intrinsics language procedure is then called with this language string. Finally, the language string is set to the return value from the language procedure. +/

2.2. Event Management

Sections in the chapter on event management should be changed or added as follows. The only change to both **XtGrabKey** and **XtGrabButton** is to add the condition that **XtUngrabKeyboard** or **XtUngrabPointer** will be called when **XFilterEvent** returns true.

2.2.1. Requesting Key and Button Grabs

In XtGrabKey add:

... the Intrinsics will call **XtUngrabKeyboard** with the timestamp from the **KeyPress** event if any of the following is true:

- There is a modal cascade and the widget is not in the active subset of the cascade and the keyboard was not previously grabbed,
- /+ **XFilterEvent** returns true. +/

And in XtGrabKey add:

... the Intrinsics will call **XtUngrabPointer** with the timestamp from the **ButtonPress** event if any of the following is true:

- There is a modal cascade and the widget is not in the active subset of the cascade and the pointer was not previously grabbed,
- /+ **XFilterEvent** returns true. +/

2.2.2. Registered Event Filters

Event filters can be registered using **XRegisterFilter**. **XtDispatchEvent** will call **XFilterEvent** before dispatching an event.

All events that would be dispatched to a widget in the absence of a filter or that have a window that does not correspond to a widget in the application context are first passed to **XFilterEvent**.

XFilterEvent will be called with the event and the window of the widget to which the Intrinsics intend to dispatch the event or the event window if **XtWindowToWidget** returns NULL.

If **XFilterEvent** returns True and the event activated a server grab, as identified by a previous call to **XtGrabKey** or **XtGrabButton**, **XtDispatchEvent** calls **XtUngrabKeyboard** or **XtUngrabPointer** with the timestamp from the event.

2.3. Resource Management

2.3.1. Resource Lists

The following entry should be added to the list of resource types.

Resource Type	Structure or Field Type

Resource Type	Structure or Field Type
XtRFontSet	FontSet

2.3.2. Predefined Resource Converters

A predefined resource converter will be added to convert from **XtRString** to **XtRFontSet**.

The string to fontset converter will act similar to the string to font converter. It will recognize the constant **XtDefaultFontSet** and evaluate this in the following manner:

- Query the resource database for the resource whose full name is ‘‘xtDefaultFontSet’’, class ‘‘XtDefaultFontSet’’ (that is, no widget name/class prefixes) and use a type **XtRString** value returned as the base font name list, or a type **XtRFontSet** value directly as the resource value.
- If the resource database does not contain a value for xtDefaultFontSet, class XtDefaultFontSet, or if a fontset cannot be created from this resource, an implementation-defined fontset is created. (One possible algorithm is to perform an **XCreateFontSet** using a wildcard base font name. This wildcard base font name should be as broad as possible to maximize the probability of locating a useable font; for example, "-*-*-*-R-*-*-*-120-*-*-*-*".)
- If no suitable fontset can be created, issue a warning message.

If a fontset was created, but missing_charset_list is not empty, a warning is issued and the partial fontset is returned.

The Intrinsics register the String to FontSet converter with a conversion argument list that extracts the current process locale at the time the converter is invoked. This insures that the converter is invoked again if the same conversion is required in a different locale.

2.3.3. StringDefs.h Header File

Add to StringDefs.h the following defines.

Resource Names:

```
#define XtNfontset "fontSet"
```

Class types:

```
#define XtCFontSet "FontSet"
```

Representation types

```
#define XtRFontSet "FontSet"
```

2.4. Utility Functions

2.4.1. Handling Errors

The source of the text is implementation dependent, there is no requirement to use the database /usr/lib/X11/XtErrorDB on POSIX-conformant systems.

2.4.2. Finding File Names

The Xt-i18n proposal proposes that **XtResolvePathname** be changed as follows to specify that the language string is in an implementation defined format.

... The substitutions specified by **XtResolvePathname** are determined from the value of the language string retrieved by **XtDisplayInitialize** for the specified display.

/- A language is specified from three parts, two of which are optional: the language, the territory, and the codeset. these parts are combined into a single string as language[_territory][.codeset]; the parts consist of arbitrary strings of letters and numbers. The underscore is omitted if territory is omitted, and the period is omitted if codeset is omitted. No interpretation of the parts of the language is done other than to use them in substitutions. -/

/+ The format and contents of the language string are implementation-defined. One suggested syntax is to compose the language string of three parts; a "language part", a "territory part" and a "codeset part". The manner in which this composition is accomplished is implementation- specific and the Intrinsics make no interpretation of the parts other than to use them in substitutions as described below. +/