USENIX Association

Proceedings of the XFree86 Technical Conference

Oakland, California, USA
November 8–9, 2001

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

XSETTINGS : A protocol for cross-toolkit settings

Owen Taylor
Red Hat, Inc.
otaylor@redhat.com

Abstract

Advancing the user experience on the open-source desktop depends upon creating standards for applications to interact with each other and the desktop environment so that applications created with different toolkits operate correctly with any desktop. The XSETTINGS protocol provides a simple example of some of the techniques and considerations that come into play for such standards. It provides a mechanism for all applications running on a desktop to have access to a common set of settings for such user-configurable parameters as double-click time and background color.

1 Introduction

Much of the functionality of a desktop depends on communications between applications on the desktop. For a long time, the only standards here were only the ones described in the X Inter-client Communication Conventions Manual [ICCCM]. Over the last several years, specifications have been developed for various additional areas of inter-client communication. For example, version 2.0 of the GTK+ library adds to the ICCCM protocols, and the Xdnd drag and drop specification [XDMD] that were supported in version 1.2, support for an extended window manager hints specification [EWMH], the XEMBED protocol for inter-application embedding and the XSETTINGS protocol for basic desktop settings [XSETTINGS].

In this paper, we'll take a detailed look at the XSETTINGS protocol. The purpose of XSETTINGS is to allow coordination of user configuration settings between applications. With full XSETTINGS support, when a setting such as double-click time or background color is changed in a desktop configuration dialog tool, it affects all applications running on that X display, no matter what widget toolkit they are using (or even if they are written using the X libraries directly without using any toolkit at all).

To make achieving this goal more feasible, XSETTINGS is restricted to a narrow set of configuration settings: desktop-wide user-configurable options that are expected to be shared between toolkits. It is not intended for application specific settings or configuration done as part of developing an application.

The traditional mechanism used for all sorts of configuration for X apps was the Xrm database [XRM]. Xrm is not restricted in its scope – it is meant to be used for desktop wide and application settings; for user settings and for developer settings. However, compared to other configuration database systems out there (such as GConf [PENNINGTON] or ACAP [RFC2244]), it has a number of limitations, for example:

- Poor support for writing configuration keys by applications or graphical configuration tools.
- No easy notification on changed settings.
- No abstraction of different configuration database backends.

For these and other reasons, Xrm is not used on modern desktops such as GNOME and KDE. However, we still need to have some way of sharing configuration settings between different toolkits. One method of achieving such shared settings would be to design a full configuration database that everybody could use. However, this would be a huge project and there would be significant barriers to adoption: it would require migration of large code bases from existing systems to the new configuration system, and the divergence of development environments between different projects (different languages, different base libraries) leaves unattractive low-level interfaces as the only ones that can be shared.

XSETTINGS takes a different approach. Instead of trying to replace existing configuration systems with a powerful replacement, it provides a common mechanism for applications running on an X display to access configuration settings without caring what the back end database is. Configuration of settings is done through the native configuration system for the desktop environment, then the the settings are presented to applications via XSETTINGS.

For XSETTINGS to achieve acceptance in this role it has to meet a number of requirements:

- Efficiency.
- Simplicity. It should be easy to implement conformant applications and toolkits, and should not require libraries beyond what is commonly deployed.
- Good facilities for change notification. It should not be necessary to restart applications for settings to take effect.
- Ability to have per-display and per-screen settings. The same settings may not make sense on every X display that a user is using.

2 Tools

The most basic question when designing XSETTINGS was what to use for a communication mechanism. Simple configuration databases (gnome-config, kde-config, libproplist), often work by storing text files in the user's home directory, removing the need for any communication mechanism with a configuration server. However, this approach doesn't allow for change notification, or per-display settings, so it won't work for our purposes. It is also problematic to store files on disk when we might want to be mirroring the GNOME configuration database on one display and the KDE configuration database for the same user logged in on a different display.

Another simple approach is to store the configuration information in a property on the root window. Properties are an X mechanism for allowing arbitrary data to be associated with a window; simply putting the configuration data on a property of the root window with a predefined name comes close to satisfying our needs. It allows for change notification (clients can choose to receive PropertyNotify events when any property on a window is changed), and for per-screen settings. This is approach (combined with on-disk files) is the approach taken by Xrm and comes close to satisfying our needs. However, it has a couple of deficiencies. First, there is no coordination of who is allowed to change the property. If there were multiple applications trying to update the property, then the config settings would behave unpredictably as first one application changed it than another. Another problem is that listening for PropertyNotify events on the root window is expensive: there are many frequently changing properties on the root window which are extraneous to the needs of a configuration mechanism. We'll see later that the actual approach that XSETTINGS takes is similar to the approach of putting the information on root window, but with a little extra sophistication. Before we look at the details of this, we'll discuss other possible forms of communication available.

In addition to properties, there are several other communications mechanism available within X: client messages allow a client to send another client an event containing a small of data (20 bytes). The selection mechanism provides for ”selections” that work as named clipboards where one client ”owns” the selection and is responsible for supplying the contents to other clients. This is typically used for selections such as the standard cut-and-paste clipboard named CLIPBOARD, or the PRIMARY selection used to hold the currently selected text.

Other mechanisms are possible that bypass the X server and communicate directly between clients. One such possibility is ICE (inter-client exchange) facility introduced in X11R6 [ICE]. This library offers a somewhat standardized way of setting up communication channels between X clients, offering facilities such as authentication and byte order negotiation. It is used as the basis of the X Session Management Protocol, and of KDE's DCOP interprocess communication protocol. However, due to various factors, such as complicated library interfaces and the lack of facilities beyond a basic communication channel, it hasn't gained much general acceptance.

Another possible communication mechanism is CORBA [CORBA]. Unlike ICE, CORBA provides a complete solution, not just a communication channel. However, it is correspondingly more complex; the specification runs to many hundreds of pages, even without covering such issues as security or the location and activation; such issues are described in separate specifications. While it is possible to identify a reasonably compact subset of CORBA that makes for a reasonable desktop-communication mechanism, there is still a hefty amount of complexity to deal with and a substantial library and set of tools. It's not really reasonable to expect different applications and toolkits on X to start using CORBA just for handling a few tasks like reading settings.

While mechanisms such as ICE and CORBA may be necessary for some complicated situations, they really are overkill for our situation, By sticking to standard X mechanisms, we make it very easy for people to implement the protocol with a minimum of complexity and requirements for dependencies beyond the core X library.

3 Architecture

The solution to both of the problems mentioned above with the root-window-property approach is something called a manager selection. As defined in the ICCCM, a manager selection is a selection used, not as a way of providing the data for a clipboard, but to provide a means of negotiating control over a unique resource. Since X needs to keep track of the owner of a selection to know which client is responsible for providing the data of the clipboard, the selection interfaces provide mechanisms for establishing and tracking ownership of a selection in a race-condition free manner. We can piggyback on top of this mechanism and use ownership of a selection as a ”lock” for ownership of some other resources. For instance, the ICCCM defines a selection manager for controlling which client acts as the window manager for a particular screen. For XSETTINGS, the we use the selection _XSETTINGS_S[N], where N is the screen number of the screen. (This allows for separate settings for each screen.) The owner of the _XSETTINGS_S[N] property is called the ”settings manager” for the screen.

The solution to the other problem mentioned above — having to select for property notify events on the root window — is provided by the fact that the owner of the selection is not actually an entire client, but just a single window. Since we store the settings data on the window owning the manager setting for XSETTINGS, clients only need to select for change notification on that window, instead of on the root window where there are lots of extraneous properties. A side benefit to this approach is that if the settings manager exits, its window is destroyed and stale configuration data doesn't stick around.

The settings manager fills the role of taking data from the desktop's native configuration database and exposing it to all applications via XSETTINGS. If live updates are desired, then the settings managers listens for changes in the configuration database by whatever means that the configuration database provides, and then updates the property on the settings manager window; a very simple settings manager could simply read a file of settings in the user's home directory, set up the property on the settings manager and then wait around until it loses the XSETTINGS selection.

To briefly summarize the conventions for the settings manager: a client that wants to provide XSETTINGS data for a screen claims ownership of the _XSETTINGS_S[N] selection by calling XSetSelectionOwner(). If there was already an owner for the selection, the client should ask for confirmation from the user before taking control of the manager selection. The window passed to XSetSelectionOwner() is a specially created unmapped window with the property data stored on it. When the settings manager loses ownership of the selection it destroys the window so that clients can detect that that the settings manager has changed and call XGetSelectionOwner() to find the new selection manager window.

The setting information is stored as a single property (_XSETTING_SETTINGS) on the selection manager window. The contents of the property is structured as a sequence of binary records for maximum simplicity and parsing speed. Another choice would have been to store the data as XML, however, this would slow down parsing considerably and also require all people listening on XSETTINGS to have a XML parser available.

There are only three setting types currently: strings, integers, and colors. While this may seem somewhat limited, anything complicated can be achieved by storing data in string setting. If you wanted to have a font as a XSETTINGS setting, you would store the font name in a string setting. A string setting could even hold XML data if a complex type was needed. In some sense, the integer and color types are just an efficiency optimization for some of the more common types of data expected to be stored in settings.

Settings are named with paths such as ”colors/background”. This allows organization of the name space to prevent conflicts. (Settings specific to a specific toolkit such as GTK+ can be given names such as GTK/colors/funky_background.) The paths aren’t significant for data storage — the data is just stored as a flat list.

To get change notification, a client selects for PropertyNotify events on the settings manager window. The easiest way for a client to do notification for individual settings to keep track of the old values for all settings, and when notification of changes is received, compare the new list with the old list, and send notifications to applications for all settings that have been added removed or changed.

An alternate facility for determining when change notification is needed for particular settings is provided by a serial number facility. The idea is that there is a serial for the entire set of settings which is updated whenever any settings has changed, and then for each setting, the serial number when it last changed is provided; to determine what settings have changed, the client only needs to keep around the global serial number from the last time it read the XSETTINGS property, and check to see if the serial number for individual properties are newer than this. However the serial number facility has not proved very useful in practice, and will likely be removed in a later version of the specification.

4 Status and Future

Sample implementations of a XSETTINGS client and settings manager are available from:

```
http://www.freedesktop.org/
  standards/xsettings.html
```

These implementations are intended to be universally usable, so they are distributed under the unrestrictive X license, and are written to use only ANSI C and the X library.

This implementation has also been incorporated into GTK+-2.0. In GTK+-2.0, XSETTINGS is used for configuring a considerable array of items, including, among other things:

- Double click time
- Cursor blink speed
- Distance threshold initiating drag-and-drop
- Widget theme

XSETTINGS is useful for GTK+ not just because it will allow GTK+ to get configuration information from all desktop environments supporting XSETTINGS but also simply to enable getting settings frmo the desktop to begin with. GTK+ has no native configuration database other than a simple mechanism for reading settings frmo a /.gtkrc file in the user’s home directory that is intended to be used when GTK+ is used without a desktop environment. In order to provide user configuration, GTK+ needs to be connected to a configuration database such as GConf via the XSETTINGS mechanism. A simple settings manager that exposes settings from GConf via XSETTINGS has been written and is available from the GNOME CVS repository. This tool will be fleshed out and GUI tools for configuring settings will be written as part of the GNOME-2.0 desktop release.

The next step in the development of XSETTINGS is to standardize on a list of standard settings names and types that can be shared across toolkits. While the XSETTINGS mechanism provides a framework for sharing user configuration, the promise of shared settings cannot be achieved until such a set of config keys is agreed upon.

Although XSETTINGS is not yet a widely adopted standard, it does provide a good example of some of the issues that come up when trying to create a cross-desktop standard. Some of the effective strategies that can be applied in this area are:

- Aim low. The simpler a proposal is to implement, and the less it effects the larger scale design of systems
- Design to work with existing systems.
- Write sample code to a lowest-common-denominator development environment (ANSI C and Xlib.)
- Discuss your proposal in appropriate forums. xdg-list@freedesktop.org (See `http://www.freedesktop.org/about/involved.html`) is a list dedicated to discussing cross-desktop integration issues for X.

5 Acknowledgments

Thanks to Waldo Bastian and Mattias Ettrich for reviewing and providing feedback on the original XSETTINGS proposal.

References

[CORBA] `http://www.corba.org`.

[EWMH] X Desktop Group *Extended Window Manager Hints*, `http://www.freedesktop.org/standards/wm-spec/` (2000).

[ICE] Robert Scheifler and Jordan Brown. *Inter-client Exchange (ICE) Protocol*, X Consortium, (1994).

[ICCCM] David Rosenthal and Stuart W. Marks, Ed. *Inter-client Communication Conventions Manual*, X Consortium (1994).

[PENNINGTON] Havoc Pennington *Introduction to the GConf Library*, `http://developer.gnome.org/feature/archive/gconf/` (2000).

[RFC2244] C. Newman and J.G. Meyers *ACAP – Application Configuration Access Protocol*, RFC 2244 (1997).

[XDMD] John Lindal *Drag-and-Drop Protocol for the X Window System*, `http://www.newplanetsoftware.com/xdnd/`

[XRM] James Gettys and Robert W. Scheifler. *Xlib – C Language X Interface*, Chapter 15. X Consortium, (1996).

[XSETTINGS] Owen Taylor *XSETTINGS - cross toolkit configuration proposal*, `http://www.freedesktop.org/standards/xsettings.html` (2001).