Application Group Extension to the X Protocol

27 September 1996
Version 1.0
X Consortium Standard
X11 Release 6.4

Kaleb S. KEITHLEY
kaleb@x.org
X Consortium, Inc.

ABSTRACT

The Application Group Extension to the X protocol is intended to provide a framework to allow more than one program to manage X applications on the desktop. The initial use of this extension will be to insert or embed the windows of X programs into the windows of another program, such as a web browser. This extension is not intended to address larger embedding issues that, for example, OpenDoc does, such as shared menu bars, etc.

Copyright © 1996 X Consortium, Inc. All Rights Reserved.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OF OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.
1. Purpose and Goals

The Application Group Extension to the X protocol is intended to provide a framework to allow more than one program to manage X applications on the desktop. The initial use of this extension will be to insert or embed the windows of X programs into the windows of another program, such as a web browser. This extension is not intended to address larger embedding issues that, for example, OpenDoc does, such as shared menu bars, etc. Using X programs on the World Wide Web allows for greater control of the presentation and takes advantage of the existing body of X programs rather than re-implement them in another language. In addition it allows the embedding of non-X programs into web browsers by using third party products like Wabi, MAE, and WinCenter.¹

¹ Wabi is a trademark of Sun Microsystems, Inc. MAE is a trademark of Apple Computer, Inc. WinCenter is a trademark of Network Computing Devices, Inc.
2. Overview of the protocol.

This extension introduces the concept of an Application Group. An Application Group is a set of one or more applications that are primarily managed by a special application known as the Application Group Leader, which, for example, might be a web browser. The primary purpose of Application Groups is to provide a means of sharing the Substructure-Redirect attribute of the root window between the window manager and one or more Application Group Leaders.

To join an Application Group an application must present the proper authorization during the connection setup. Authorizations are generated by the X server at the request of an Application Group Leader, and are then stored for the application to use to establish its connection to the X server. To generate an authorization the Application Group Leader sends a request to the server naming the Application Group to which the authorization will be bound, and any applications that connect using that authorization will automatically become part of the associated Application Group. The protocol to generate an authorization is defined in the Security Extension specification.

As a member of an Application Group, when an application creates and maps a window as a child of the root window, the MapRequest and ConfigureRequest events are delivered to the Application Group Leader instead of the window manager. The Application Group Leader may then reparent the window into its own window hierarchy; or reissue the map request, in which case the window comes under the control of the window manager.
3. Requests

AppGroupQueryVersion

\[
\begin{align*}
\text{client_major_version}: &\text{ CARD16} \\
\text{client_minor_version}: &\text{ CARD16} \\
=> \\
\text{server_major_version}: &\text{ CARD16} \\
\text{server_minor_version}: &\text{ CARD16}
\end{align*}
\]

If supplied, the \textit{client_major_version} and \textit{client_minor_version} indicate what version of the protocol the application wants the server to implement. The server version numbers returned indicate the version of the protocol the X server actually supports. This may not match the versions requested by the application. An implementation may (but need not) support more than one version simultaneously. The \textit{server_major_version} and \textit{server_minor_version} numbers are a mechanism to support any future revisions of the Application Group extension protocol which may be necessary. In general, the major version would increment for incompatible changes, and the minor version would increment for small, upward-compatible changes. X servers that support the protocol defined in this document will return a \textit{server_major_version} of 1 and a \textit{server_minor_version} of 0.

AppGroupCreate

\[
\begin{align*}
\text{app_group}: &\text{ APPGROUP} \\
\text{value_mask}: &\text{ BITMASK} \\
\text{value_list}: &\text{ LISTofVALUE}
\end{align*}
\]

This request creates an Application Group using \textit{app_group} as the Application Group ID.

The \textit{value_mask} and \textit{value_list} specify attributes of the Application Group that are to be explicitly initialized. The attributes, their types, and the default values are:

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
Attribute & Type & Default \\
\hline
\text{app_group_leader} & \text{ Bool} & \text{ True} \\
\hline
\text{single_screen} & \text{ Bool} & \text{ True} \\
\hline
\text{default_root} & \text{ Window} & \text{ None} \\
\hline
\text{root_visual} & \text{ VisualID} & \text{ None} \\
\hline
\text{default_colormap} & \text{ Colormap} & \text{ None} \\
\hline
\text{black_pixel} & \text{ Pixel} & \text{ 0} \\
\hline
\text{white_pixel} & \text{ Pixel} & \text{ 0} \\
\hline
\end{tabular}
\end{center}
If the `single_screen` attribute is True then the number of video screens returned to a program in the Application Group in the connection setup message is one, irrespective of how many video screens the server actually has. If a server supports both video and print screens, then all print screens will always be returned. If `single_screen` is specified as `True` then the connection setup message will contain only the information about the video screen which has `default_root` as its root window, plus any print screens.

The intent is to allow an embedding manager to ensure that it will be able to reparent any top-level windows that Application Group members create. By hiding the fact that there are other screens it can be reasonably assured that applications will only create top-level windows on the same screen that it itself appears on. An embedding manager should take care not to supply an invalid display, e.g. :0.1, to a program that will be in an Application Group where the `single_screen` attribute is True.

If `single_screen` is set to True `default_root` specifies which screen will be returned as screen zero in the connection setup message for applications in the Application Group. If set to None, then the real screen zero is used, otherwise the screen which has `default_root` as its root window will be used.

If `single_screen` is set to True the `root_visual` and `default_colormap` attributes may be used to over-ride the default values that are returned in the connection setup information returned to new programs in the Application Group. If None is specified for `root_visual` or `default_colormap` then the normal default values for the screen (possibly specified by `default_root`) are used, otherwise the specified values are used. If `root_visual` and/or `default_colormap` are specified they must be valid, i.e. `root_visual` must be a visual type available on the screen, and the colormap, if specified, must be a valid colormap for the visual that is used.

IF `single_screen` is set to True and `default_colormap` is not specified as None, the `black_pixel` and `white_pixel` attributes must be specified, and they will over-ride the default values that are returned in the connection setup returned to new programs in the Application Group. If `default_colormap` is specified as None and `black_pixel` and/or `white_pixel` are specified, they will be ignored.

The `app_group_leader` attribute is used to identify the Application Group Leader program for the `app_group`. By specifying True the server will identify the program making the request as the Application Group Leader for the application group. The Application Group Leader receives MapRequest and ConfigureRequest events from the server when an attempt is made to map or configure top-level windows of a program in an Application Group, instead of being sent to a window manager that has selected SubstructureRedirect events on the root window. The parent window field in these events will contain the Application Group ID.

**AppGroupDestroy**

    app_group: APPGROUP

This request destroys the `app_group`. If the `app_group_leader` attribute for the `app_group` is True, then any applications in the Application Group that are still connected will be killed as if a
KillClient request had been received for that application.

If the application that created a non-embedded Application Group exits, and therefore any Authorizations to be cancelled, and any applications that attempt to open new connections to the X server using one of those Authorizations will be unable to do so.

**AppGroupGetAttr**

\[
app\_group: \text{APPGROUP} \\
\Rightarrow \text{LISTofVALUE}
\]

This request returns the application group attributes for \( app\_group \).

**AppGroupQuery**

\[
\text{resource: XID} \\
\Rightarrow \text{app\_group: APPGROUP}
\]

This request returns the Application Group ID of the application that created \( resource \) or None if that application is not associated with any Application Group. The \( resource \) value may be the resource base of the application.

**AppGroupCreateAssociation**

\[
\text{window: WINDOW} \\
\text{window\_type: CARD32} \\
\text{system\_window: LISTofCARD8}
\]

This request associates \( window \) with \( system\_window \). The \( window\_type \) indicates the native window system of the application making the request. For non-X \( window\_types \) both the embedding manager and the server must be executing on the same host. When \( system\_window \) is Microsoft Windows or OS/2 Presentation Manager, the \( system\_window \) is an HWND; when the native window system is Macintosh, the \( system\_window \) is a WindowPtr and a Rect. The \( window \) may be used for any X request that takes a Window.

**AppGroupDestroyAssociation**

\[
\text{window: WINDOW}
\]

This request destroys the association created with AppGroupCreateAssociation. The \( window \) is destroyed. The \( system\_window \) that was specified in the AppGroupCreateAssociation request is
not affected.
4. Changes to Existing Requests

MapWindow

If the override-redirect attribute of the window is False and if the window is a child of a root window and if the window belongs to an application that is in an application group and if some other application is the application group leader for that group, then a MapRequest event is generated and the window remains unmapped. Otherwise, the core protocol semantics apply.

ConfigureWindow

If the override-redirect attribute of the window is False and if the window is a child of a root window and if the window belongs to an application that is in an application group and if some other application is the application group leader for that group, then a ConfigureRequest event is generated and the window remains unchanged. Otherwise, the core protocol semantics apply.

CreateWindow

When a program in an Application Group creates a window that is a child of a root window and specifies CopyFromParent for the Visual, if the single_screen attribute is True and the root_visual attribute is set to something other than None, then the window will be created using the Application Group’s root_visual, otherwise core protocol semantics apply.

When a program in an Application Group creates a window that is a child of a root window and specifies CopyFromParent for the Colormap, if the single_screen attribute is True, the default_colormap attribute is set to something other than None, and the window’s Visual is the same as the Application Group’s root_visual attribute, then the window will be created using the Application Group’s default_colormap, otherwise core protocol semantics apply.

ChangeWindowAttributes

When a program in an Application Group changes the attributes of a window that is a child of a root window and specifies CopyFromParent for the Colormap, if the single_screen attribute is True, the default_colormap attribute is set to something other than None, and the window’s Visual is the same as the Application Group’s root_visual attribute, then the window will be created using the Application Group’s default_colormap, otherwise core protocol semantics apply.
5. Changes to Existing Events

When the top-level window of an application that is a member of an Application Group is the target of a MapWindow or ConfigureWindow request, if there is an Application Group Leader then MapRequest and ConfigureRequest events are automatically delivered to it, otherwise the core protocol semantics apply, i.e. they are delivered to the client, if any, that has SubstructureRedirect set in its root-window event mask, e.g. the window manager.

The Application Group Leader must not select SubstructureRedirect events on a root window as doing so would result in a core protocol error; only one client is permitted to do so, and that is usually the window manager.

MapRequest

When a MapWindow request is received for a window whose override-redirect attribute is set to False and whose parent is the root window and the window belongs to an application that is in an application group and there is an application group leader for the group, then this event is delivered to the Application Group Leader with the parent field in the event set to the AppGroup ID. Otherwise the core protocol semantics apply.

ConfigureRequest

When a ConfigureWindow request is received for a window whose override-redirect attribute is set to False and whose parent is the root window and the window belongs to an application that is in an application group and there is an application group leader for the group, then this event is delivered to the Application Group Leader with the parent field in the event set to the AppGroup ID. Otherwise the core protocol semantics apply.
6. Errors

**AppGroupQueryVersion**

There are no errors for AppGroupQueryVersion.

**AppGroupCreate**

A Window error is returned if *default_root* is specified and is not a valid root window.

A Color error is returned if *default_colormap* is specified but *default_colormap* is not a valid colormap for the screen of *default_root*.

A Match error is returned if *root_visual* and *default_colormap* are both specified, but *default_colormap*’s visual is not *root_visual*.

A Match error is returned if *root_visual* does not exist for the screen of the *default_root*.

**AppGroupDestroy**

An AppGroup error is returned if *app_group* is not a valid Application Group.

An Access error is returned if an untrusted application attempts to destroy an Application Group created by a trusted application.

**AppGroupGetAttr**

An AppGroup error is returned if *app_group* is not a valid Application Group.

An Access error is returned if an untrusted application attempts to get the attributes of an Application Group created by a trusted application.

**AppGroupQuery**

An Access error is returned if an untrusted application attempts to query the Application Group of a trusted application.

**AppGroupCreateAssociation**

A Match error is returned if the X server does not support the *window_type*.

An Access error may be returned if the X server only supports the *window_type* on the local host.
and the program making the request is on a non-local host.

A Window error may be returned for system-specific errors related to `system_window`, e.g. `system_window` does not represent a valid native window.

**AppGroupDestroyAssociation**

A Window error is returned if `window` was not specified in a previous `AppGroupCreateAssociation` request.
7. Encoding

Please refer to the X11 Protocol encoding document as this document uses conventions established there.

The name of this extension is XC-APPGROUP

**AppGroupQueryVersion**

```plaintext
1  CARD8  opcode
1  0  XC-APPGROUP opcode
2  3  length
2  CARD16  client_major_version
2  CARD16  client_minor_version

=>
1  1  Reply
1  unused
2  CARD16  sequence_number
4  0  length
2  CARD16  server_major_version
2  CARD16  server_minor_version
20  unused
```

**AppGroupCreate**

```plaintext
1  CARD8  opcode
1  1  XC-APPGROUP opcode
2  8+n  length
4  XID  app_group
4  BITMASK  attrib_mask
  #x00000001  app_group_leader
  #x00000002  single_screen
  #0x00000004  default_root
  #x00000008  root_visual
  #x00000010  default_colormap
  #x00000020  black_pixel
  #x00000040  white_pixel
n  LISTofVALUE  value-list
```

VALUES
```plaintext
  4  BOOL  app_group_leader
  4  BOOL  single_screen
  4  WINDOW  default_root
  4  VISUALID  root_visual
  4  COLOORMAP  default_colormap
  4  CARD32  black_pixel
  4  CARD32  white_pixel
```
AppGroupDestroy
  1  CARD8 opcode
  1  2 XC-APPGROUP opcode
  2  2 length
  4  XID app_group

AAppGroupGetAttr
  1  CARD8 opcode
  1  4 XC-APPGROUP opcode
  2  2 length
  4  XID app_group
=>
  1  1 Reply
  1 unused
  2  CARD16 sequence_number
  4  0 length
  4  WINDOW default_root
  4  VISUALID root_visual
  4  COLORMAP default_colormap
  4  CARD32 black_pixel
  4  CARD32 white_pixel
  1  BOOL single_screen
  1  BOOL app_group_leader
  2 unused

AppGroupQuery
  1  CARD8 opcode
  1  5 XC-APPGROUP opcode
  2  2 length
  4  XID resource
=>
  1  1 Reply
  1 unused
  2  CARD16 sequence_number
  4  0 length
  4  XID app_group
  20 unused

AppGroupCreateAssoc
  1  CARD8 opcode
  1  6 XC-APPGROUP opcode
2 n length
4 WINDOW window
2 CARD16 window_type
   #0 X11
   #1 Macintosh
   #2 Win32, OS/2 PM 2.x
   #3 Win16, OS/2 PM 1.x
2 n system_window_len
n LISTofCARD8 system_window

AppGroupDestroyAssoc
1 CARD8 opcode
1 7 XC-APPGROUP opcode
2 2 length
4 WINDOW window
8. Library API

Status XagQueryVersion(
    Display* dpy,
    int* major_version_return,
    int* minor_version_return);

XagQueryVersion sets major_version_return and minor_version_return to the major and minor Application Group protocol version supported by the server. If the Xag library is compatible with the version returned by the server it returns non-zero. If dpy does not support the Application Group extension, or if the server and library protocol versions are incompatible, or if there was an error during communication with the server, it returns zero. No other Xag functions may be called before this function. If a program violates this rule, the effects of all subsequent Xag calls that it makes are undefined.

An embedding manager in, e.g. a Personal Computer Web Browser, will need to open a connection to the Personal Computer X server by calling XOpenDisplay() before using the Application Group extension.

An embedding manager such as a web browser that intends to embed programs in an Application Group should create the Application Group with XagCreateEmbeddedApplicationGroup.

Status XagCreateEmbeddedApplicationGroup(
    Display* dpy,
    VisualID root_visual,
    Colormap default_colormap,
    unsigned long black_pixel,
    unsigned long white_pixel,
    XAppGroup* app_group_return);

XagCreateEmbeddedApplicationGroup creates an Application Group for an embedding manager with the attributes specified. It also sets the default_root attribute to DefaultRoot(dpy, DefaultsScreen(dpy)) and the single_screen and app_group_leader attributes to True. It returns the Application Group ID in app_group_return.

You can create an Application Group without intending to do embedding. One reason for doing this is to give a group of clients their own font-path.

A special font-path can be created by creating an Application Group, getting an Authorization using XSecurityGenerateAuthorization, and then running ‘xset fp+ <new font path>’ as a member of the Application Group. Font-path elements added in this way will be “private” to the Application Group.

Status XagCreateNonembeddedApplicationGroup(
    Display* dpy,
    XAppGroup* app_group_return);
An Application Group created with XagCreateNonembdeddedApplicationGroup will have the default_root, root_visual, and default_colormap attributes all set to None; the single_screen and app_group_leader attributes are set to False, and the black_pixel and white_pixel attributes are not used since the default_colormap attribute is None.

To destroy an Application Group use XagDestroyApplicationGroup.

Status XagDestroyApplicationGroup(
    Display* dpy,
    XAppGroup app_group);

The Application Group specified by app_group is destroyed. If the Application Group was created using XagCreateEmbeddingApplicationGroup, i.e. and therefore the app_group_leader attribute is True, all programs that are members of the Application Group are killed as if a KillClient request had been issued.

To retrieve the attributes of an Application Group use XagGetApplicationGroupAttributes.

Status XagGetApplicationGroupAttributes(
    Display* dpy,
    XAppGroup app_group,
    ...);

XagGetApplicationGroupAttributes is a varargs function that retrieves the Application Group’s attributes specified in the vararg parameter list.

The attributes that may be specified are: XagNappGroupLeader, XagNsingleScreen, XagNdefaultRoot, XagNrootVisual, XagNdefaultColormap, XagNblackPixel, and XagNwhitePixel; which correspond to app_group_leader, single_screen, default_root, root_visual, default_colormap, black_pixel, and white_pixel respectively. See AppGroupCreate in Section 3 for a description of each attribute.

The types for each of the parameters are pointers to the following:

    single_screen    Bool
    default_root     Window
    root_visual      VisualID
    default_colormap Colormap
    black_pixel      unsigned long
    white_pixel      unsigned long
    app_group_leader Bool

Example:
... Boolean app_group_leader, single_screen;
Window default_root;
VisualID root_visual;
Colormap default_colormap;
Pixel black_pixel, white_pixel;
...
status = XagGetApplicationGroupAttributes(dpy, app_group,
   XagNappGroupLeader, &app_group_leader,
   XagNsingleScreen, &single_screen,
   XagNdefault_root, &default_root,
   XagNrootVisual, &root_visual,
   XagNdefaultColormap, &default_colormap,
   XagNblackPixel, &black_pixel,
   XagNwhitePixel, &white_pixel,
   NULL);
...

To determine which Application Group a resource (such as a window) belongs to, use XagQueryApplicationGroup.

Status XagQueryApplicationGroup(  
   Display* dpy,
   XID resource,
   XAppGroup* app_group_return);

The Application Group is returned in app_group_return, if the resource is not in any Application Group then app_group_return will be set to None.

To associate an X Window ID with a system-specific window ID, such as a HWND or a WindowPtr, use XagCreateAssociation.

Status XagCreateAssociation(  
   Display* dpy,
   Window* window_return,
   void* system_window);

The window_return may be used as the target for a ReparentWindow request.

Because XReparentWindow is not constrained in the same way that Win32’s SetParent and the Macintosh are, there is no reason to call XagCreateAssociation in an X-based embedding manager. As such if XagCreateAssociation is called in a native X program, the window_return will be the same as the system_window, and the implementation may even elect to not generate any protocol.

To create an association on the Macintosh:
struct {
    WindowPtr win;
    Rect rect;
} system_window;

system_window.win = win_ptr;
system_window.rect.top = system_window.rect.left = 20;
system_window.rect.bottom = 180;
system_window.rect.right = 380;

status = XagCreateAssociation (dpy, &window, (void*)&system_window);

To create an association using a Win16, Win32, or OS/2 PM:

HWND system_window;

status = XagCreateAssociation (dpy, &window, (void*)&system_window);

To destroy the association created with XagCreateAssociation use XagDestroyAssociation.

Status XagDestroyAssociation(
    Display* dpy,
    Window window);

After calling XagDestroyAssociation the window may no longer be used to reparent windows with XReparentWindow.

Like XagCreateAssociation, if the native window system is X11 the implementation may elect to not generate any protocol as a result of this function call in order to avoid unintentionally destroying the the system_window that was specified in the prior XagCreateAssociation call.
APPENDIX A: System Window Encodings

The AppGroupCreateAssoc request has the following possible variations:

**AppGroupCreateAssoc (X11)**

1. CARD8  opcode
2. 6  XC-APPGROUP opcode
2. n  length
4. WINDOW  window
2. 0  window_type
2. 4  system_window_len
4. WINDOW  Window

**AppGroupCreateAssoc (Macintosh)**

1. CARD8  opcode
2. 6  XC-APPGROUP opcode
2. n  length
4. WINDOW  window
2. 1  window_type
2. 12  system_window_len
4. CARD32  WindowPtr
2. INT16  Rect.top
2. INT16  Rect.left
2. INT16  Rect.bottom
2. INT16  Rect.right

**AppGroupCreateAssoc (Win32)**

1. CARD8  opcode
2. 6  XC-APPGROUP opcode
2. n  length
4. WINDOW  window
2. 2  window_type
2. 4  system_window_len
4. CARD32  HWND

**AppGroupCreateAssoc (Win16)**

1. CARD8  opcode
2. 6  XC-APPGROUP opcode
2. n  length
4. WINDOW  window
2. 3  window_type
2. 4  system_window_len
2. CARD16  HWND offset
2. CARD16  HWND segment