Desktop Notifications Specification

Mike Hearn

mike@navi.cx

Christian Hammond

chipx86@chipx86.com

Version 0.9

Revision History

Revision 0.9 15 January 2006 Revised by: cdh
Clarify the naming for the application IDs. Put back a number of things that probably shouldn't have been removed from the spec.
Revision 0.8 23 September 2005 Revised by: J5
Major overhaul of spec to work with the newer D-Bus recursive type system. Simplify protocol. Changed the verbage notification type to category
Revision 0.7 28 July 2005 Revised by: cdh
Added "x" and "y" hints. Talk about the variant type for hint values.
Revision 0.6 1 April 2005 Revised by: cdh
Updated to work with D-BUS 0.31+.
Revision 0.5 2 October 2004 Revised by: cdh
Added a "suppress-sound" hint. Added a "sound" capability. Renamed the "soundfile" hint to sound-file".
Revision 0.4 29 September 2004 Revised by: cdh
Added image support in markup, and made the restrictions on markup more clear. Removed the High urgency. Added new notification types. Fixed notification expiration.
Revision 0.3 15 September 2004 Revised by: cdh
Added hint and notification type sections
Revision 0.2 foo Revised by: mh
Added replaces field to protocol
Revision 0.1 foo Revised by: mh
Initial version

1. Introduction

This is a draft standard for a desktop notifications service, through which applications can generate passive popups (sometimes known as "poptarts") to notify the user in an asynchronous manner of events.

This specification explicitly does not include other types of notification presentation such as modal message boxes, window manager decorations or window list annotations.

Example use cases include:

- Presence changes in IM programs: for instance, MSN Messenger on Windows pioneered the use of passive popups to indicate presence changes.
- Scheduled alarm
- Completed file transfer
- New mail notification
- Low disk space/battery warnings

2. Basic Design

In order to ensure that multiple notifications can easily be displayed at once, and to provide a convenient implementation, all notifications are controlled by a single session-scoped service which exposes a D-BUS interface.

On startup, a conforming implementation should take the `org.freedesktop.Notifications` service on the session bus. This service will be referred to as the "notification server" or just "the server" in this document. It can optionally be activated automatically by the bus process, however this is not required and notification server clients must not assume that it is available.

The server should implement the `org.freedesktop.Notifications` interface on an object with the path `"/org/freedesktop/Notifications"`. This is the only interface required by this version of the specification.

A notification has the following components:

Table 1. Notification Components

Component	Description
Application Name	This is the optional name of the application sending the notification. This should be the application's formal name, rather than some sort of ID. An example would be "FredApp E-Mail Client," rather than "fredapp-email-client."
Replaces ID	An optional ID of an existing notification that this notification is intended to replace.
Notification Icon	The notification icon. This is represented either as a URI (file:// is the only URI schema supported right now) or a name in a freedesktop.org-compliant icon theme (not a GTK+ stock ID).

Component	Description
Summary	This is a single line overview of the notification. For instance, "You have mail" or "A friend has come online". It should generally not be longer than 40 characters, though this is not a requirement, and server implementations should word wrap if necessary. The summary must be encoded using UTF-8.
Body	This is a multi-line body of text. Each line is a paragraph, server implementations are free to word wrap them as they see fit. The body may contain simple markup as specified in Markup. It must be encoded using UTF-8. If the body is omitted, just the summary is displayed.
Actions	The actions send a request message back to the notification client when invoked. This functionality may not be implemented by the notification server, conforming clients should check if it is available before using it (see the GetCapabilities message in Protocol). An implementation is free to ignore any requested by the client. As an example one possible rendering of actions would be as buttons in the notification popup. Actions are sent over as a list of pairs. Each even element in the list (starting at index 0) represents the identifier for the action. Each odd element in the list is the localized string that will be displayed to the user. The default action (usually invoked my clicking the notification) should have a key named `"default"`. The name can be anything, though implementations are free not to display it.

Component	Description
Hints	See Hints. Beyond the core protocol is the hints table. A couple of core elements have been moved to hints mostly because in a huge number of cases their default values would be sufficent. The elements moved to hints are: **Elements Moved to Hints** **Element:** Category ID **Description:** An optional ID representing the type of notification (the name has been changed from Notification Type ID in pervious versions). See Categories. **Element:** Urgency Level **Description:** The urgency of the notification. See Urgency Levels. (Defaults to 1 - Normal) **Element:** Icon Data **Description:** Instead of overloading the icon field we now have an icon_data field that is used when icon is blank.
Expiration Timeout	The timeout time in milliseconds since the display of the notification at which the notification should automatically close. If -1, the notification's expiration time is dependent on the notification server's settings, and may vary for the type of notification. If 0, the notification never expires.

Each notification displayed is allocated a unique ID by the server. This is unique within the session. While the notification server is running, the ID will not be recycled unless the capacity of a uint32 is exceeded.

This can be used to hide the notification before the expiration timeout is reached. It can also be used to atomically replace the notification with another. This allows you to (for instance) modify the contents of a notification while it's on-screen.

3. Backwards Compatibility

Clients should try and avoid making assumptions about the presentation and abilities of the notification server. The message content is the most important thing.

Clients can check with the server what capabilities are supported using the `GetCapabilities` message. See Protocol.

If a client requires a response from a passive popup, it should be coded such that a non-focus-stealing message box can be used in the case that the notification server does not support this feature.

4. Markup

Body text may contain markup. The markup is XML-based, and consists of a small subset of HTML along with a few additional tags.

The following tags should be supported by the notification server. Though it is optional, it is recommended. Notification servers that do not support these tags should filter them out.

 ... 	Bold
<i> ... </i>	Italic
<u> ... </u>	Underline
 ... 	Hyperlink
	Image

A full-blown HTML implementation is not required of this spec, and notifications should never take advantage of tags that are not listed above. As notifications are not a substitute for web browsers or complex dialogs, advanced layout is not necessary, and may in fact limit the number of systems that notification services can run on, due to memory usage and screen space. Such examples are PDAs, certain cell phones, and slow PCs or laptops with little memory.

For the same reason, a full XML or XHTML implementation using XSLT or CSS stylesheets is not part of this specification. Information that must be presented in a more complex form should use an application-specific dialog, a web browser, or some other display mechanism.

The tags specified above mark up the content in a way that allows them to be stripped out on some implementations without impacting the actual content.

4.1. Hyperlinks

Hyperlinks allow for linking one or more words to a URI. There is no requirement to allow for images to be linked, and it is highly suggested that implementations do not allow this, as there is no clean-looking, standard visual indicator for a hyperlinked image.

Hyperlinked text should appear in the standard blue underline format.

Hyperlinks cannot function as a replacement for actions. They are used to link to local directories or remote sites using standard URI schemes.

Implementations are not required to support hyperlinks.

4.2. Images

Images may be placed in the notification, but this should be done with caution. The image should never exceed 200x100, but this should be thought of as a maximum size. Images should always have alternative text provided through the `alt="..."` attribute.

Image data cannot be embedded in the message itself. Images referenced must always be local files.

Implementations are not required to support images.

5. Icons

A notification can optionally have an icon specified by the Notification Icon field or by the icon_data hint.

The icon_data field should be a raw image data structure of signature (iiibiiay) which describes the width, height, rowstride, has alpha, bits per sample, channels and image data respectively.

6. Categories

Notifications can optionally have a type indicator. Although neither client or nor server must support this, some may choose to. Those servers implementing categories may use them to intelligently display the notification in a certain way, or group notifications of similar types.

Categories are in `class.specific` form. `class` specifies the generic type of notification, and `specific` specifies the more specific type of notification.

If a specific type of notification does not exist for your notification, but the generic kind does, a notification of type `class` is acceptable.

Third parties, when defining their own categories, should discuss the possibility of standardizing on the hint with other parties, preferably in a place such as the xdg (http://freedesktop.org/mailman/listinfo/xdg) mailing list at freedesktop.org (http://freedesktop.org/). If it warrants a standard, it will be added to the table above. If no consensus is reached, the category should be in the form of "`x-vendor.class.name`."

The following table lists standard notifications as defined by this spec. More will be added in time.

Table 2. Categories

Type	Description
`"device"`	A generic device-related notification that doesn't fit into any other category.
`"device.added"`	A device, such as a USB device, was added to the system.
`"device.error"`	A device had some kind of error.
`"device.removed"`	A device, such as a USB device, was removed from the system.
`"email"`	A generic e-mail-related notification that doesn't fit into any other category.
`"email.arrived"`	A new e-mail notification.
`"email.bounced"`	A notification stating that an e-mail has bounced.
`"im"`	A generic instant message-related notification that doesn't fit into any other category.
`"im.error"`	An instant message error notification.
`"im.received"`	A received instant message notification.
`"network"`	A generic network notification that doesn't fit into any other category.
`"network.connected"`	A network connection notification, such as successful sign-on to a network service. This should not be confused with `device.added` for new network devices.
`"network.disconnected"`	A network disconnected notification. This should not be confused with `device.removed` for disconnected network devices.
`"network.error"`	A network-related or connection-related error.

Type	Description
`"presence"`	A generic presence change notification that doesn't fit into any other category, such as going away or idle.
`"presence.offline"`	An offline presence change notification.
`"presence.online"`	An online presence change notification.
`"transfer"`	A generic file transfer or download notification that doesn't fit into any other category.
`"transfer.complete"`	A file transfer or download complete notification.
`"transfer.error"`	A file transfer or download error.

7. Urgency Levels

Notifications have an urgency level associated with them. This defines the importance of the notification. For example, "Joe Bob signed on" would be a low urgency. "You have new mail" or "A USB device was unplugged" would be a normal urgency. "Your computer is on fire" would be a critical urgency.

Urgency levels are defined as follows:

Table 3. Urgency Levels

Type	Description
0	Low
1	Normal
2	Critical

Developers must use their own judgement when deciding the urgency of a notification. Typically, if the majority of programs are using the same level for a specific type of urgency, other applications should follow them.

For low and normal urgencies, server implementations may display the notifications how they choose. They should, however, have a sane expiration timeout dependent on the urgency level.

Critical notifications should not automatically expire, as they are things that the user will most likely want to know about. They should only be closed when the user dismisses them, for example, by clicking on the notification.

8. Hints

Hints are a way to provide extra data to a notification server that the server may be able to make use of.

Neither clients nor notification servers are required to support any hints. Both sides should assume that hints are not passed, and should ignore any hints they do not understand.

Third parties, when defining their own hints, should discuss the possibility of standardizing on the hint with other parties, preferably in a place such as the xdg (http://freedesktop.org/mailman/listinfo/xdg) mailing list at freedesktop.org (http://freedesktop.org/). If it warrants a standard, it will be added to the table above. If no consensus is reached, the hint name should be in the form of `"x-*vendor*-*name*."`

The value type for the hint dictionary in D-BUS is of the `DBUS_TYPE_VARIANT` container type. This allows different data types (string, integer, boolean, etc.) to be used for hints. When adding a dictionary of hints, this type must be used, rather than putting the actual hint value in as the dictionary value.

The following table lists the standard hints as defined by this specification. Future hints may be proposed and added to this list over time. Once again, implementations are not required to support these.

Table 4. Standard Hints

Name	Value Type	Description
`"urgency"`	byte	The urgency level.
`"category"`	string	The type of notification this is.
`"desktop-entry">`	string	This specifies the name of the desktop filename representing the calling program. This should be the same as the prefix used for the application’s .desktop file. An example would be "rhythmbox" from "rhythmbox.desktop". This can be used by the daemon to retrieve the correct icon for the application, for logging purposes, etc.
`"image_data"`	(iiibiiay)	This is a raw data image format which describes the width, height, rowstride, has alpha, bits per sample, channels and image data respectively. We use this value if the icon field is left blank.
`"sound-file"`	string	The path to a sound file to play when the notification pops up.

Name	Value Type	Description
`"suppress-sound"`	boolean	Causes the server to suppress playing any sounds, if it has that ability. This is usually set when the client itself is going to play its own sound.
`"x"`	int	Specifies the X location on the screen that the notification should point to. The `"y"` hint must also be specified.
`"y"`	int	Specifies the Y location on the screen that the notification should point to. The `"x"` hint must also be specified.

9. D-BUS Protocol

The following messages *must* be supported by all implementations.

9.1. Message commands

9.1.1. `org.freedesktop.Notifications.GetCapabilities`

```
STRING_ARRAY org.freedesktop.Notifications.GetCapabilities (void);
```

This message takes no parameters.

It returns an array of strings. Each string describes an optional capability implemented by the server. The following values are defined by this spec:

Table 5. Server Capabilities

`"actions"`	The server will provide the specified actions to the user. Even if this cap is missing, actions may still be specified by the client, however the server is free to ignore them.
`"body"`	Supports body text. Some implementations may only show the summary (for instance, onscreen displays, marquee/scrollers)

"body-hyperlinks"	The server supports hyperlinks in the notifications.
"body-images"	The server supports images in the notifications.
"body-markup"	Supports markup in the body text. If marked up text is sent to a server that does not give this cap, the markup will show through as regular text so must be stripped clientside.
"icon-multi"	The server will render an animation of all the frames in a given image array. The client may still specify multiple frames even if this cap and/or "icon-static" is missing, however the server is free to ignore them and use only the primary frame.
"icon-static"	Supports display of exactly 1 frame of any given image array. This value is mutually exclusive with "icon-multi", it is a protocol error for the server to specify both.
"sound"	The server supports sounds on notifications. If returned, the server must support the "sound-file" and "suppress-sound" hints.

New vendor-specific caps may be specified as long as they start with `"x-`*`vendor`*`"`. For instance, `"x-gnome-foo-cap"`. Capability names must not contain spaces. They are limited to alpha-numeric characters and dashes (`"-"`).

9.1.2. `org.freedesktop.Notifications.Notify`

```
UINT32 org.freedesktop.Notifications.Notify (STRING app_name, UINT32
replaces_id, STRING app_icon, STRING summary, STRING body, ARRAY actions,
DICT hints, INT32 expire_timeout);
```

Sends a notification to the notification server.

Table 6. Notify Parameters

Name	Type	Description
app_name	STRING	The optional name of the application sending the notification. Can be blank.

Name	Type	Description
replaces_id	UINT32	The optional notification ID that this notification replaces. The server must atomically (ie with no flicker or other visual cues) replace the given notification with this one. This allows clients to effectively modify the notification while it's active. A value of value of 0 means that this notification won't replace any existing notifications.
app_icon	STRING	The optional program icon of the calling application. See Icons. Can be an empty string, indicating no icon.
summary	STRING	The summary text briefly describing the notification.
body	STRING	The optional detailed body text. Can be empty.
actions	ARRAY	Actions are sent over as a list of pairs. Each even element in the list (starting at index 0) represents the identifier for the action. Each odd element in the list is the localized string that will be displayed to the user.
hints	DICT	Optional hints that can be passed to the server from the client program. Although clients and servers should never assume each other supports any specific hints, they can be used to pass along information, such as the process PID or window ID, that the server may be able to make use of. See Hints. Can be empty.

Name	Type	Description
expire_timeout	INT32	The timeout time in milliseconds since the display of the notification at which the notification should automatically close. If -1, the notification's expiration time is dependent on the notification server's settings, and may vary for the type of notification. If 0, never expire.

If *replaces_id* is 0, the return value is a UINT32 that represent the notification. It is unique, and will not be reused unless a `MAXINT` number of notifications have been generated. An acceptable implementation may just use an incrementing counter for the ID. The returned ID is always greater than zero. Servers must make sure not to return zero as an ID.

If *replaces_id* is not 0, the returned value is the same value as *replaces_id*.

9.1.3. `org.freedesktop.Notifications.CloseNotification`

```
void org.freedesktop.Notifications.CloseNotification (UINT32 id);
```

Causes a notification to be forcefully closed and removed from the user's view. It can be used, for example, in the event that what the notification pertains to is no longer relevant, or to cancel a notification with no expiration time.

The `NotificationClosed` signal is emitted by this method.

If the notification no longer exists, an empty D-BUS Error message is sent back.

9.1.4. `org.freedesktop.Notifications.GetServerInformation`

```
void org.freedesktop.Notifications.GetServerInformation (out STRING name, out
STRING vendor, out STRING version);
```

This message returns the information on the server. Specifically, the server name, vendor, and version number.

Table 7. GetServerInformation Return Values

Name	Type	Description
name	STRING	The product name of the server.
vendor	STRING	The vendor name. For example, "KDE," "GNOME," "freedesktop.org," or "Microsoft."
version	STRING	The server's version number.

9.2. Signals

9.2.1. `org.freedesktop.Notifications.NotificationClosed`

`org.freedesktop.Notifications.NotificationClosed` `(UINT32 id, UINT32 reason);`

A completed notification is one that has timed out, or has been dismissed by the user.

Table 8. NotificationClosed Parameters

Name	Type	Description
id	UINT32	The ID of the notification that was closed.
reason	UINT32	The reason the notification was closed. 1 - The notification expired. 2 - The notification was dismissed by the user. 3 - The notification was closed by a call to `CloseNotification`. 4 - Undefined/reserved reasons.

The ID specified in the signal is invalidated *before* the signal is sent and may not be used in any further communications with the server.

9.2.2. `org.freedesktop.Notifications.ActionInvoked`

`org.freedesktop.Notifications.ActionInvoked` `(UINT32` *`id`*`, STRING` *`action_key`*`);`

This signal is emitted when one of the following occurs:

- The user performs some global "invoking" action upon a notification. For instance, clicking somewhere on the notification itself.
- The user invokes a specific action as specified in the original Notify request. For example, clicking on an action button.

Table 9. ActionInvoked Parameters

Name	Type	Description
id	UINT32	The ID of the notification emitting the ActionInvoked signal.
action_key	STRING	The key of the action invoked. These match the keys sent over in the list of actions.

> **Note:** Clients should not assume the server will generate this signal. Some servers may not support user interaction at all, or may not support the concept of being able to "invoke" a notification.