Desktop Notifications Specification

Mike Hearn

mike@navi.cx

Christian Hammond

chipx86@chipx86.com

Table of Contents

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

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:

Application Name

: 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.

Application Icon

: An optional byte array containing the application's icon. This should be in PNG or GIF formats.

Application ID

: An optional byte array containing the application's icon. This should be in PNG or GIF formats.

Replaces ID

: An optional ID of an existing notification that this notification is intended to replace.

Notification Type ID

: An optional ID representing the notification type. See Notification Types.

Urgency Level

The urgency of the notification. See Urgency Levels.

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 text 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.

Images

See Icons and Sounds.

Sounds

See Icons and Sounds.

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.

Hints

See Hints.

Expiration Time

The timestamp in seconds since the epoch that the notification should close. For example, if one wishes to have an expiration of 5 seconds from now, they must grab the current timestamp and add 5 seconds to it.

If zero, the notification's expiration time is dependent on the notification server's settings, and may vary for the type of notification.

The expiration time should be respected by implementations, but this is not required (this is for compatibility with KNotify).

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 time 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.

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 instead and the notification server is only used when available.

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.

`<b> ... </b>`	Bold
`<i> ... </i>`	Italic
`<u> ... </u>`	Underline
`<a href="..."> ... </a>`	Hyperlink

* *What else do we want here? We're going to want more tags for sure.*

Icons and Sounds

A notification can optionally include an array of images and/or a single sound. The array of images specifies frames in an animation, animations always loop. Implementations are free to ignore the images and sound data, and implementations that support images need not support animation.

If the image array has more than one element, a "primary frame" can be specified - if not specified it defaults to the first frame. For implementations that support images but not animation (for instance a `KNotify` bridge), only the primary frame will be used.

Each element of the array must have the same type as the first element, mixtures of strings and blobs are not allowed. The element types can be one of the following:

Element	Type	Description
Icon Theme Name	String	Any string that does not begin with the / character is assumed to be an icon theme name and is looked up according to the spec. The best size to fit the servers chosen presentation will be used. This is the recommended way of specifying images.

Element	Type	Description
Absolute Path	String	Any string that begins with a / will be used as an absolute file path. Implementations should support at minimum files of type image/png and image/svg.
Image Data	Binary Data	A data stream may be embedded in the message. This is assumed to be of type image/png.

A sound can be specified, this will be played by the notification server when the notification is displayed.

* *Elaborate here!*

Notification Types

* *Write me!*

Urgency Levels

* *Write me!*

Hints

* *Write me!*

D-BUS Protocol

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

Message commands

`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:

"body"	Supports body text. Some implementations may only show the summary (for instance, onscreen displays, marquee/scrollers)
"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.
"static-image"	Supports display of exactly 1 frame of any given image array. This value is mutually exclusive with "multi-image", it is a protocol error for the server to specify both.
"multi-image"	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 static-image is missing, however the server is free to ignore them and use only the primary frame.
"sound"	The server will play the specified sound. Even if this cap is missing, a sound may still be specified however the server is free to ignore it.
"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.

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

org.freedesktop.Notifications.Notify

```
UINT32 org.freedesktop.Notifications.Notify (STRING_OR_NIL app_name,
BYTE_ARRAY_OR_NIL app_icon, UINT32_OR_NIL replaces_id, STRING_OR_NIL
notification_type, BYTE urgency_level, STRING summary, STRING_OR_NIL
body, ARRAY images, STRING_OR_NIL sound, DICT_OR_NIL actions,
DICT_OR_NIL hints, UINT32_OR_NIL expire_time);
```

Sends a notification to the notification server.

Table 1. Notify Parameters

Name	Type	Description
app_name	STRING or NIL	The optional name of the application sending the notification.

Name	Type	Description
app_icon	BYTE_ARRAY or NIL	The optional program icon of the calling application.
replaces_id	UINT32 or NIL	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.
notification_type	STRING or NIL	The optional notification type ID, for potential server categorization and logging purposes. See Notification Types.
urgency_level	BYTE	The urgency level. See Urgency Levels.
summary	STRING	The summary text briefly describing the notification.
body	STRING or NIL	The optional detailed body text.
images	ARRAY or NIL	The optional array of images. See Icons and Sounds.
sound	STRING	The optional sound file to play.
actions	DICT or NIL	A dictionary key of actions. Each key is the localized name of the action, as it should appear to the user, and maps to a UINT32 value containing a program-specific action code. This code will be reported back to the program if the action is invoked by the user.
hints	DICT or NIL	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.

Name	Type	Description
expire_time	UINT32 or NIL	The notification time-out time, represented as UNIX-time (seconds since the epoch). If this is NIL, the notification will never time out, and will only be closed when an action is invoked. If non-NIL, this will specify a time at which the notification will be automatically closed. If zero, the server's default expiration time will be used.

If *replaces_id* is NIL, 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 NIL, the returned value is the same value as *replaces_id*.

org.freedesktop.Notifications.CloseNotification

```
BOOLEAN 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 onger relevant, or to cancel a notification with no expiration time.

The `NotificationClosed` signal is emitted by this method.

This returns `TRUE` if the notification was closed. If the notification didn't exist, this will return `FALSE`. This can happen if the notification already expired or was closed, or if the ID was simply invalid.

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 2. GetServerInformation Return Values

Name	Type	Description
name	STRING	The product name of the server.

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

Signals

`org.freedesktop.Notifications.NotificationClosed`

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

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

Table 3. NotificationClosed Return Values

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 - "Other" 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.