The GNOME configuration files

Miguel de Icaza

miguel@gnu.org

GNOME applications and the GNOME libraries use a number of special files to configure the GNOME desktop. By dropping new files in special locations or customizing these files the GNOME desktop can be configured in a number of ways.

MIME type information

MIME type handling in the GNOME desktop

MIME types are using to distinguish what sort of operations and attributes a file has. The operations are usually things like: how to open files of a given mime-type, how to view them, what icon is used to display it and so on.

Besides the MIME types, in GNOME, the metadata system is used to provide information in a per-file basis (for example, to override the defaults of the mime-type).

GNOME uses a number of techniques depending on the speed requirements to figure out what the MIME type of a file is. It is possible to get the mime-type by only using the filename or if speed is not important, by checking the actual file contents (for more information look at the gnome-mime.h header file and the accompanying documentation).

There are ways in GNOME to fetch the values bound to a mime-type with the routines provided in the gnome-libs routine `gnome-mime-info.c`.

Mapping a filename to a MIME type.

The GNOME system uses MIME types to identify the contents of files. In addition, the GNOME system can bind arbitrary information to these MIME types by the Keys system. These are used within the system to associate actions, icons, and various operations to MIME types, and by extension to the files on the system

To make the GNOME desktop aware of a new MIME type it is necessary to install a file with the extension `.mime` in the `$gnome/share/mime-info` directory or from the `~/.gnome/mime-info directory.`

The file `$gnome/share/mime-info/gnome.mime` is special, as it contains the defaults for gnome, and is read first. In addition, the file `~/.gnome/mime-info/user.mime` is read last. This will guarantee that there is a way to set system defaults, and there is a way for the user to override them. There is currently no way to tell anything about the order of the other files in those directories, nor is there anyway to override system defaults yet.

The format is the following:

Example 1. Definining a MIME type

```
mime-type-name
ext[,priority]: ext1 ext2 ext3
ext[,priority]: ext4
regex[,priority]: regex1
regex[,priority]: regex2
```

where `"mime-type-name"` is a valid MIME type. For example *"text/plain"*.

Multiple ext and regex entries can be provided for this mime-type. Please note that the regex keyword only allows one regular expression per line, while the ext keyword lets you list a number of extensions in the same line.

The priority indicates the number of components that your extension matches (the bigger the number, the better). Right now, only priorities 1 and 2 are supported.

This is required to distinguish the mime type correctly for situations like `emacs.tar.gz`, where the `.gz` extension could match the "compressed" rule, while the real match should be for `"tar.gz"`, so a `"tar.gz"` would have a higher priority

For example, for a vCalendar application, this file would be installed:

Example 2. `calendar.mime` sample file

```
application/v-calendar:
ext: vcf
```

Adding keys to a MIME type

To add keys to a mime-type, it is necessary to install a file with the extension `.keys` in the `$gnome/share/mime-info` directory or from the `~/.gnome/mime-info` directory.

Similar to the `.mime` files, there is a provision for a `$gnome/share/mime-info/gnome.keys` file and a `~/.gnome/mime-info/user.keys` file. This, as above, allows for both defaults and user preferences.

The full content of the .keys database is the sum of all of the .keys files in the directory.

The file has the following format:

Example 3. Sample keys definition file

```
mime-type-match:
[\[LANG\]]key=value
```

"mime-type-match" allows you to provide either a full mime-type, or a glob expression, for example, you can list `"image/gif"` for making the entries apply to the image/gif mime type or `"image/*"` if you want the keys to apply to all of the images.

The [LANG] part is optional, and it used to provide internationalized versions of any strings appearing in the value.

key is the key (a number of keys are standardized) and value is the value bound to it.

Given that the actual keys definition can come from multiple sources, it is possible for example to ship a file that only defines icons for a set of well known files and format (this is the case for the file manager) and have specific applications install the actual properties when they are installed

For example, the GIMP could ship with this file:

Example 4. Sample GIMP keys file

```
image/*:
open=gimp %f

image/x-xcf:
icon-filename=/opt/gimp/share/xcf.png
```

This will make the GIMP the handler for the open action. Files of type xcf would use the filename pointed in the icon-filename key.

%f gets interpolated with the file name or the list of file names that matched this mime-type.

As you can see, a `.keys` file does not need to provide all of the values, it can just provide or override some of the actions.

Note that user defined bindings in `.keys` file will take precedence over system installed files.

Default keys

The following keys are recognized in the GNOME desktop:

Table 1. Standard keys used by the GNOME desktop

Key name	Action
icon-filename	points to a filename with the icon that should be used to represent files of this type
open	open the give file with this command.
fm-open	file-manager open. If present, the file manager will use this action instead of the value in open to perform this action (the file manager for example will open archive files as if they were directories by using the VFS).
view	Command to view the file contents.
fm-view	file-manager view. If present, invoking the view operation on the file manager will use the value defined here instead of the value in "view".
ascii-view	A command that should be used to do an ascii-rendering of the file. Used as a fall-back by the filemanager if a "view" action does not exist.
fm-ascii-view	Fall-back operation for the file manager as well.

Those keys are also queried on the metadata (except in the cases where the lookup would be too expensive).

Defining multiple actions

Sometimes it is important to provide more than one "open" action. To do this, the keys should be specified in the following fashion (The same applies to any other key that might need more than one option):

Example 5. Multiple open actions defined

```
text/plain:
open=command %f
open.tag1.Do this and that=this-and-that %f
```

```
open.tag2.Gzip it=gzip %f
open.tag3.Mail it=mail %{Recipient} < %f
```

This will define a default action (which would invoke "command %f"), and three extra open commands:

Table 2. Sample mappings

Open key name	Mapping
Do this and that	this-and-that %f
Gzip it	gzip %f
Mail it	mail ${Recipient} < %f

Please notice that you have to add a "tag" to every "open" key.

Localizing the texts for a tagged key

The reason for using the tags, is that they are required to distinguish the correct value for internationalization purposes, here is how the above example would be translated to Spanish:

Example 6. Sample keys with localization support

```
text/plain:
open=command %f
open.tag1.Do this and that=this-and-that %f
[es]open.tag1.Hacer esto y aquello=this and that %f
open.tag2.Gzip it=gzip %f
[es]open.tag2.Comprimir con gzip=gzip %f
open.tag3.Mail it=mail %{Recipient} < %f
[es]open.tag3.Enviarlo por correo=mail %{Recipient} < %f
open.tag4.Example=example %f
```

To the application programmer, if the code is running with LANG=es, he will get the following keys: "open", "open.tag1.Hacer esto y aquello", "open.tag2.Comprimir con gzip", "open.tag3.Enviarlo por correo" and "open.tag4.Example"

If a translation entry were missing, he would get the general definition for the default value (in the example above, I did not translate the value for open.tag4.

Help files in GNOME

The GNOME libraries provide a special macro to include a quick index to the help files installed by a package. To achieve this, the application should install its help files and the `topic.dat`in a special directory that the GNOME libraries will search at runtime.

The directory is the `$(datadir)/gnome/help/CODEID/LOCALE/`. Where CODEID is the CODEID used in the GnomeUIInfo structure for help files. And LOCALE is the locale code name. The english documentation will set the LOCALE to C.

In this directory, a number of HTML files are expected to be installed (The GNOME project uses Docbook for its documentation, and it installs their documentation there).

The special file `topic.dat` is used by the `GnomeApp` helper routines to construct the index of available help for the application. The format of this file is very simple: it consists of a number of lines, one per topic, and the first string up to the first tab character is the filename (might include anchors) for the topic being defined, then the tab character followed by text that will appear on the menu (what the user will see).

For example, here is the topic.dat for the GNOME calendar application for the C locale is:

Example 7. GNOME calendar `topic.dat` file

```
gnomecal.html   Manual
cmdline.html    Command line options
```

We suggest that users create in their programs a `doc` directory and that they create a per-locale directory. This will allow translators to install their own translated versions of the work in there with little effort.

Users of Automake can use this fragment for installing the help files in the proper location:

Example 8. Automake fragment for installing GNOME help files

```
package_helpdir = $(datadir)/gnome/help/package/C

package_help_DATA =     \
        package.html    \
        topic.dat
```

GNOME Object Activation Directory

CORBA Server Handling in the GNOME Desktop

To allow applications to request access to a specific CORBA object, GNOME CORBA servers place information in GOAD, the GNOME Object Activation Directory. This directory stores information on the CORBA objects that a program can provide to other programs. Each directory entry contains a unique implementation identifier (the "GOAD ID"), a list of interfaces that the object supports, a human-readable description of the object implementation, and information on how to create a new instance of the object implementation.

If an application provides the implementation for a CORBA object, it is simple to integrate that object into the GOAD system. An application would install a '.goad' data file into the correct directory as part of its installation process. Then, a few simple function calls must be made when the object is created and destroyed, and the application must handle an extra command line option.

Included with the GNOME libraries is 'goad-browser'. This program lists all the objects registered with GOAD, and allows the user to activate these objects, and see which objects are currently running.

Complex activation arrangements are also possible with GOAD. An object factory can be used to arbitrate the object creation process in a manner that is transparent to the client. CORBA implementations in shared libraries also can be loaded directly into the address space of the application, facilitating higher performance.

GNOME Desktop Sound Configuration

Allowing Configurable Per-Application Sound Events

GNOME applications can easily register events for which they want user-configurable sounds to be played, and then trigger those events. This is done through the gnome triggers system.

The first step in adding sound events is to create a .soundlist file that describes the application and each event. The format is the following:

Example 9. Defining an application's sound event

```
[__section_info__]
description=MyApplication

[event1]
file=default1.wav
description=This is a sound event

[event2]
file=default2.wav
description=This is another event
```

The description in the '__section_info__' section should be the name of the application as the user is to see it. Then, a section for each event provides a description, and the filename of the default sound to play.

The application will also need to call the gnome_triggers_do function with the appropriate parameters whenever the event in question occurs. By following these two steps, the application will allow the user to easily configure what sounds are assigned to important application events.

File type conversion configuration files

GNOME includes a number of built-in routines to convert files from different formats by using rules. These routines are used by a few programs in the system. A number of conversion rules are provided as part of the default GNOME installation, and it is possible for users and system administrators to extend these rules with their own rules and their own filters to improve the system.

The system uses MIME types to identify the source file format and the target file format. The GNOME routines will try to convert the files from the source to the target with the mininum number of operations.

The file format conversion is achieved by invoking a number of filters that convert their standard input from one mime type to another mime type.

The rules for converting from a mime-type to another mime-type are loaded from the `$gnome-prefix/share/type-convert` directory, any files in that directory whose extension is `.convert` is loaded as well as the per-user `~/.gnome/type.convert` file.

The file format used to define a filter is very simple:

Example 10. Sample MIME type file conversion definition

```
image/gif image/x-portable-pixmap giftopnm
```

This sample file conversion file defines a conversion rule from the image/gif MIME type to the image/x-portable-pixmap MIME type. The program used for the conversion is `giftopnm`. This program should do the specified MIME type conversion by reading its standard input and writing the results on the standard output.

Configuring the desktop

It is possible to configure the desktop for the users on your system. This is accomplished by populating the `$libdir/mc/desktop-scripts` directory.

The directory $(libdir)/mc/desktop-scripts is a place holder for information on how to configure the default's user desktop. There are a number of files that can be installed in this directory:

- Link definition files: These files are used to create links to web and ftp sites on the user's desktop.
- Executables: Programs or links to programs that will be executed when the user requests a directory rescan.

Link definition file format

The file should contain an ini-like file whose contents are as follows:

Example 11. Sample link definition file

```
[linkname]
title=Title in english
title[NN]=Title in language NN
type=url
url=some-url
```

The linkname is the name of the file that will be created in the user's desktop directory.

title is the caption that will be used by this file. This can be localized by providing one or more title[NN] entries, where NN is the setting that would be set by the user for the NN locale.

Executables

Executable programs will be executed like this:

```
program -desktop-dir DESKTOPDIR
```

Where DESKTOPDIR is the user's desktop directory.

The program can then populate the directory with any information it might want. This is used for example by the g-print program (that provides priting facilities by creating special desktop entries that set up the drop action and various different open actions)