Desktop Menu Specification

Waldo Bastian

<waldo.bastian@intel.com>

Francois Gouget

<fgouget@codeweavers.com>

Alex Graveley

<alex@ximian.com>

George Lebl

<jirka@5z.com>

Havoc Pennington

<hp@pobox.com>

Heinrich Wendel

<h_wendel@cojobo.net>

Version 1.0

Table of Contents

Introduction

This document defines how to construct a user-visible hierarchy of applications, typically displayed as a menu. It allows third-party software to add menu items that work for all desktops, and allows system administrators to edit menus in a way that affects all desktops.

The basic scheme is very simple. Information about each application (menu item) is stored in a desktop entry (see Desktop Entry Standard). Then an XML configuration file defines the hierarchical arrangement (layout) of menu items, and which menu items are actually displayed.

Things are complicated somewhat by the need to support legacy desktop entry hierarchies, and the need to allow third parties to extend the menu layout. Both of these issues are addressed by the idea of *merging* two menu layouts.

In addition to a strict definition of the contents of each menu this specification also foresees in a number of layout / presentation hints. This part of the specification is optional, implementations may chose to ignore these hints.

File locations

Files involved in this specification are located according to the "desktop base directory specification".

Here are the files defined by this specification:

`$XDG_CONFIG_DIRS`/menus/`${XDG_MENU_PREFIX}`applications.menu

> This file contains the XML definition of the main application menu layout. The first file found in the search path should be used; other files are ignored. This implies that if the user has their own `${XDG_MENU_PREFIX}`applications.menu, it replaces the system wide one. (Though the user's menu may explicitly merge the

system wide one.)

Systems that offer multiple desktop environments and that want to use distinct menu layouts in the different environments can use differently prefixed .menu files. In this case the `$XDG_MENU_PREFIX` environment variable must be set by the system to reflect the .menu file that is being used.

For example if a system contains both the GNOME and the KDE desktop environments it can decide to use gnome-applications.menu as the menu layout in GNOME sessions and kde-applications.menu as the menu layout in KDE sessions. To correctly reflect this, it should set the `$XDG_MENU_PREFIX` environment variable to "gnome-" respectively "kde-".

Implementations may chose to use .menu files with other names for tasks or menus other than the main application menu. Such usage is not covered by this specification.

`$XDG_CONFIG_DIRS`/menus/applications-merged/

The default merge directories included in the <DefaultMergeDirs> element. By convention, third parties may add new <Menu> files in this location to create their own sub-menus.

Note that a system that uses either gnome-applications.menu or kde-applications.menu depending on the desktop environment in use must still use applications-merged as the default merge directory in both cases.

Implementations may chose to use .menu files with names other than application.menu for tasks or menus other than the main application menu. In that case the first part of the name of the default merge directory is derived from the name of the .menu file.

For example in a system that uses a preferences.menu file to describe an additional menu, the default merge directories included in the <DefaultMergeDirs> element in the preferences.menu file would become `$XDG_CONFIG_DIRS`/menus/preferences-merged/

`$XDG_DATA_DIRS`/applications/

This directory contains a .desktop file for each possible menu item. Each directory in the `$XDG_DATA_DIRS` search path should be used (i.e. desktop entries are collected from all of them, not just the first one that exists). When two desktop entries have the same name, the one appearing earlier in the path is used.

The <DefaultAppDirs> element in a menu file indicates that this default list of desktop entry locations should be scanned at that point. If a menu file does not contain <DefaultAppDirs>, then these locations are not scanned.

`$XDG_DATA_DIRS`/desktop-directories/

This directory contains directory entries which may be associated with folders in the menu layout. Each directory in the search path should be used. Only files ending in .directory are used; other files are ignored.

The <DefaultDirectoryDirs> element in a menu file indicates that this default list of directory entry locations should be scanned at that point. If a menu file does not contain <DefaultDirectoryDirs>, then these locations are not scanned.

Extensions to the desktop entry format

This specification adds three new fields to desktop entries: `Categories`, `OnlyShowIn` and `NotShowIn`.

The `Categories` field is a list of strings used to classify menu items. For example, applications in the `AudioVideo` category might end up in the "Sound & Video" submenu. Appendix A, *Registered Categories* enumerates the standard categories. Categories not in this document must be prefixed by the string "X-" indicating that they are extensions. Categories are case-sensitive.

Desktop entries should list all categories that clearly apply. They should not list categories that only vaguely or possibly apply, because the user will end up seeing the same desktop entry in a half-dozen places. But it's typical that several categories will apply to a given desktop entry.

The `OnlyShowIn` field is a list of strings identifying the environments that should display a given menu item. If an `OnlyShowIn` field is present, a given environment should only display the menu item if the string identifying that environment is present. The strings are case-sensitive. Appendix B, *Registered OnlyShowIn Environments* enumerates the strings to use for some common environments.

The `NotShowIn` field is a list of strings identifying the environments that should not display a given menu item. If an `NotShowIn` field is present, a given environment should only display the menu item if the string identifying that environment is not present. The strings are case-sensitive. Appendix B, *Registered OnlyShowIn Environments* enumerates the strings to use for some common environments.

Examples of using `Categories` and `OnlyShowIn`

A desktop entry for a Qt-based image viewer might contain this `Categories` line:

```
Categories=Qt;Graphics;RasterGraphics;Viewer;
```

A desktop entry for Octave, a command-line mathematics program (which would also have the field `Terminal=true`), might have:

```
Categories=ConsoleOnly;Math;
```

A desktop entry for a GNOME-specific calculator program that should only appear in GNOME might have:

```
Categories=GNOME;Utility;
OnlyShowIn=GNOME;
```

Note that the `OnlyShowIn` field is a *list* and thus ends in a semicolon.

Format of menu files

Menu files must be well-formed XML files and end in the extension ".menu". They should also conform to the menu file DTD which implies that implementation-specific extensions to the file format are not allowed. Implementations may stop processing if they encounter a menu file which does not comply with the associated DTD. Note that the associated DTD may differ in version from the one defined in this document.

When an implementation updates an existing menu file it may need to update the identifier to a newer version of the DTD. Implementations should never update the identifier of an existing menu file to an older version. In order to remain compatible with newer versions, implementations should ignore and preserve any XML elements, attributes and attribute values that it does not know how to handle.

Document Type Declaration

Menu files for this version of the specification must use the following namespace, public and system identifiers:

Namespace

```
http://www.freedesktop.org/standards/menu
```

Public Identifier for 1.0

```
PUBLIC "-//freedesktop//DTD Menu 1.0//EN"
```

System Identifier for 1.0

```
http://www.freedesktop.org/standards/menu-spec/menu-1.0.dtd
```

Here is a sample document type declaration:

```
<!DOCTYPE Menu PUBLIC "-//freedesktop//DTD Menu 1.0//EN"
"http://www.freedesktop.org/standards/menu-spec/menu-1.0.dtd">
```

All menu files MUST include the document type declaration, so that implementations can adapt to different versions of this specification (and so implementations can validate the menu file against the DTD).

Elements

<Menu>

The root element is <Menu>. Each <Menu> element may contain any number of nested <Menu> elements, indicating submenus.

<AppDir>

This element may only appear below <Menu>. The content of this element is a directory name. Desktop entries in this directory are scanned and added to the pool of entries which can be included in this <Menu> and its submenus. Only

files ending in ".desktop" should be used, other files are ignored.

Desktop entries in the pool of available entries are identified by their *desktop-file id* (see Desktop-File Id). The desktop-file id of a desktop entry is equal to its filename, with any path components removed. So given a <AppDir> `/foo/bar` and desktop entry `/foo/bar/Hello.desktop` the desktop entry would get a desktop-file id of `Hello.desktop`

If the directory contains sub-directories then these sub-directories should be (recursively) scanned as well. The name of the subdirectory should be added as prefix to the desktop-file id together with a dash character ("-") So given a <AppDir> `/foo/bar` and desktop entry `/foo/bar/booz/Hello.desktop` the desktop entry would get a desktop-file id of `booz-Hello.desktop` A desktop entry `/foo/bar/bo/oz/Hello.desktop` would result in a desktop-file id of `bo-oz-Hello.desktop`

<AppDir> elements appearing later in the menu file have priority in case of collisions between desktop-file ids.

If the filename given as an <AppDir> is not an absolute path, it should be located relative to the location of the menu file being parsed.

Duplicate <AppDir> elements (that specify the same directory) should be ignored, but the *last* duplicate in the file should be used when establishing the order in which to scan the directories. This is important when merging (see the section called “Merging”). The order of <AppDir> elements with respect to <Include> and <Exclude> elements is not relevant, also to facilitate merging.

<DefaultAppDirs>

This element may only appear below <Menu>. The element has no content. The element should be treated as if it were a list of <AppDir> elements containing the default app dir locations (*`datadir`*/applications/ etc.). When expanding <DefaultAppDirs> to a list of <AppDir>, the default locations that are earlier in the search path go later in the <Menu> so that they have priority.

<DirectoryDir>

This element may only appear below <Menu>. The content of this element is a directory name. Each directory listed in a <DirectoryDir> element will be searched for directory entries to be used when resolving the <Directory> element for this menu and its submenus. If the filename given as a <DirectoryDir> is not an absolute path, it should be located relative to the location of the menu file being parsed.

Directory entries in the pool of available entries are identified by their *relative path* (see Relative path).

If two directory entries have duplicate relative paths, the one from the last (furthest down) element in the menu file must be used. Only files ending in the extension ".directory" should be loaded, other files should be ignored.

Duplicate <DirectoryDir> elements (that specify the same directory) are handled as with duplicate <AppDir> elements (the last duplicate is used).

<DefaultDirectoryDirs>

This element may only appear below <Menu>. The element has no content. The element should be treated as if it were a list of <DirectoryDir> elements containing the default desktop dir locations (*datadir*/desktop-directories/ etc.). The default locations that are earlier in the search path go later in the <Menu> so that they have priority.

<Name>

Each <Menu> element must have a single <Name> element. The content of the <Name> element is a name to be used when referring to the given menu. Each submenu of a given <Menu> must have a unique name. <Menu> elements can thus be referenced by a menu path, for example "Applications/Graphics." The <Name> field must not contain the slash character ("/"); implementations should discard any name containing a slash. See also Menu path.

<Directory>

Each <Menu> element has any number of <Directory> elements. The content of the <Directory> element is the relative path of a directory entry containing meta information about the <Menu>, such as its icon and localized name. If no <Directory> is specified for a <Menu>, its <Name> field should be used as the user-visible name of the menu.

Duplicate <Directory> elements are allowed in order to simplify menu merging, and allow user menus to override system menus. The last <Directory> element to appear in the menu file "wins" and other elements are ignored, unless the last element points to a nonexistent directory entry, in which case the previous element should be tried instead, and so on.

<OnlyUnallocated> and <NotOnlyUnallocated>

Each <Menu> may contain any number of <OnlyUnallocated> and <NotOnlyUnallocated> elements. Only the last such element to appear is relevant, as it determines whether the <Menu> can contain any desktop entries, or only those desktop entries that do not match other menus. If neither <OnlyUnallocated> nor <NotOnlyUnallocated> elements are present, the default is <NotOnlyUnallocated>.

To handle <OnlyUnallocated>, the menu file must be analyzed in two conceptual passes. The first pass processes <Menu> elements that can match any desktop entry. During this pass, each desktop entry is marked as allocated according to whether it was matched by an <Include> rule in some <Menu>. The second pass processes only <Menu> elements that are restricted to unallocated desktop entries. During the second pass, queries may only match desktop entries that were not marked as allocated during the first pass. See the section called “Generating the menus”.

<Deleted> and <NotDeleted>

Each <Menu> may contain any number of <Deleted> and <NotDeleted> elements. Only the last such element to appear is relevant, as it determines

whether the <Menu> has been deleted. If neither <Deleted> nor <NotDeleted> elements are present, the default is <NotDeleted>. The purpose of this element is to support menu editing. If a menu contains a <Deleted> element not followed by a <NotDeleted> element, that menu should be ignored.

<Include>

An <Include> element is a set of rules attempting to match some of the known desktop entries. The <Include> element contains a list of any number of matching rules. Matching rules are specified using the elements <And>, <Or>, <Not>, <All>, <Filename>, and <Category>. Each rule in a list of rules has a logical OR relationship, that is, desktop entries which match any rule are included in the menu.

<Include> elements must appear immediately under <Menu> elements. The desktop entries they match are included in the menu. <Include> and <Exclude> elements for a given <Menu> are processed in order, with queries earlier in the file handled first. This has implications for merging, see the section called “Merging”. See the section called “Generating the menus” for full details on how to process <Include> and <Exclude> elements.

<Exclude>

Any number of <Exclude> elements may appear below a <Menu> element. The content of an <Exclude> element is a list of matching rules, just as with an <Include>. However, the desktop entries matched are removed from the list of desktop entries included so far. (Thus an <Exclude> element that appears before any <Include> elements will have no effect, for example, as no desktop entries have been included yet.)

<Filename>

The <Filename> element is the most basic matching rule. It matches a desktop entry if the desktop entry has the given desktop-file id. See Desktop-File Id.

<Category>

The <Category> element is another basic matching predicate. It matches a desktop entry if the desktop entry has the given category in its `Categories` field.

<All>

The <All> element is a matching rule that matches all desktop entries.

<And>

The <And> element contains a list of matching rules. If each of the matching rules inside the <And> element match a desktop entry, then the entire <And> rule matches the desktop entry.

<Or>

The <Or> element contains a list of matching rules. If any of the matching rules

inside the <Or> element match a desktop entry, then the entire <Or> rule matches the desktop entry.

<Not>

The <Not> element contains a list of matching rules. If any of the matching rules inside the <Not> element matches a desktop entry, then the entire <Not> rule does *not* match the desktop entry. That is, matching rules below <Not> have a logical OR relationship.

<MergeFile [type="path"|"parent"] >

Any number of <MergeFile> elements may be listed below a <Menu> element, giving the name of another menu file to be merged into this one. the section called “Merging” specifies how merging is done. The root <Menu> of the merged file will be merged into the immediate parent of the <MergeFile> element. The <Name> element of the root <Menu> of the merged file are ignored.

If the type attribute is missing or set to "path" then the contents of the <MergeFile> element indicates the file to be merged. If this is not an absolute path then the file to be merged should be located relative to the location of the menu file that contains this <MergeFile> element.

Duplicate <MergeFile> elements (that specify the same file) are handled as with duplicate <AppDir> elements (the last duplicate is used).

If the type attribute is set to "parent" and the file that contains this <MergeFile> element is located under one of the paths specified by `$XDG_CONFIG_DIRS`, the contents of the element should be ignored and the remaining paths specified by `$XDG_CONFIG_DIRS` are searched for a file with the same relative filename. The first file encountered should be merged. There should be no merging at all if no matching file is found.

Compatibility note: The filename specified inside the <MergeFile> element should be ignored if the type attribute is set to "parent", it should however be expected that implementations based on previous versions of this specification will ignore the type attribute and that such implementations will use the filename inside the <MergeFile> element instead.

Example 1: If `$XDG_CONFIG_HOME` is "~/.config/" and `$XDG_CONFIG_DIRS` is "/opt/gnome /:/etc/xdg/" and the file ~/.config/menus/applications.menu contains <MergeFile type="parent">/opt/kde3/etc/xdg/menus/applications.menu</MergeFile> then the file /opt/gnome/menus/applications.menu should be merged if it exists. If that file does not exists then the file /etc/xdg/menus/applications.menu should be merged instead.

Example 2: If `$XDG_CONFIG_HOME` is "~/.config/" and `$XDG_CONFIG_DIRS` is "/opt/gnome /:/etc/xdg/" and the file /opt/gnome/menus/applications.menu contains <MergeFile type="parent">/opt/kde3/etc/xdg/menus /applications.menu</MergeFile> then the file /etc/xdg/menus/applications.menu should be merged if it exists.

<MergeDir>

Any number of <MergeDir> elements may be listed below a <Menu> element. A <MergeDir> contains the name of a directory. Each file in the given directory which ends in the ".menu" extension should be merged in the same way that a <MergeFile> would be. If the filename given as a <MergeDir> is not an absolute path, it should be located relative to the location of the menu file being parsed. The files inside the merged directory are not merged in any specified order.

Duplicate <MergeDir> elements (that specify the same directory) are handled as with duplicate <AppDir> elements (the last duplicate is used).

<DefaultMergeDirs>

This element may only appear below <Menu>. The element has no content. The element should be treated as if it were a list of <MergeDir> elements containing the default merge directory locations. When expanding <DefaultMergeDirs> to a list of <MergeDir>, the default locations that are earlier in the search path go later in the <Menu> so that they have priority.

<LegacyDir>

This element may only appear below <Menu>. The text content of this element is a directory name. Each directory listed in a <LegacyDir> element will be an old-style legacy hierarchy of desktop entries, see the section called “Legacy Menu Hierarchies” for how to load such a hierarchy. Implementations must not load legacy hierarchies that are not explicitly specified in the menu file (because for example the menu file may not be the main menu). If the filename given as a <LegacyDir> is not an absolute path, it should be located relative to the location of the menu file being parsed.

Duplicate <LegacyDir> elements (that specify the same directory) are handled as with duplicate <AppDir> elements (the last duplicate is used).

The <LegacyDir> element may have one attribute, `prefix`. Normally, given a <LegacyDir> `/foo/bar` and desktop entry `/foo/bar/baz/Hello.desktop` the desktop entry would get a desktop-file id of `Hello.desktop`. Given a prefix of `boo-`, it would instead be assigned the desktop-file id `boo-Hello.desktop`. The prefix should not contain path separator ('/') characters.

<KDELegacyDirs>

This element may only appear below <Menu>. The element has no content. The element should be treated as if it were a list of <LegacyDir> elements containing the traditional desktop file locations supported by KDE with a hard coded prefix of "kde-". When expanding <KDELegacyDirs> to a list of <LegacyDir>, the locations that are earlier in the search path go later in the <Menu> so that they have priority. The search path can be obtained by running `kde-config --path apps`

<Move>

This element may only appear below <Menu>. The <Move> element contains pairs of <Old>/<New> elements indicating how to rename a descendant of the current <Menu>. If the destination path already exists, the moved menu is merged with the destination menu (see the section called “Merging” for details).

<Move> is used primarily to fix up legacy directories. For example, say you are merging a <LegacyDir> with folder names that don't match the current hierarchy; the legacy folder names can be moved to the new names, where they will be merged with the new folders.

<Move> is also useful for implementing menu editing, see the section called “Menu editing”.

<Old>

This element may only appear below <Move>, and must be followed by a <New> element. The content of both <Old> and <New> should be a menu path (slash-separated concatenation of <Name> fields, see Menu path). Paths are interpreted relative to the menu containing the <Move> element.

<New>

This element may only appear below <Move>, and must be preceded by an <Old> element. The <New> element specifies the new path for the preceding <Old> element.

<Layout>

The <Layout> element is an optional part of this specification. Implementations that do not support the <Layout> element should preserve any <Layout> elements and their contents as far as possible. Each <Menu> may optionally contain a <Layout> element. If multiple elements appear then only the last such element is relevant. The purpose of this element is to offer suggestions for the presentation of the menu. If a menu does not contain a <Layout> element or if it contains an empty <Layout> element then the default layout should be used. The <Layout> element may contain <Filename>, <Menuname>, <Separator> and <Merge> elements. The <Layout> element defines a suggested layout for the menu starting from top to bottom. References to desktop entries that are not contained in this menu as defined by the <Include> and <Exclude> elements should be ignored. References to sub-menus that are not directly contained in this menu as defined by the <Menu> elements should be ignored.

<DefaultLayout [show_empty="false"] [inline="false"] [inline_limit="4"] [inline_header="true"] [inline_alias="false"]>

The <DefaultLayout> element is an optional part of this specification. Implementations that do not support the <DefaultLayout> element should preserve any <DefaultLayout> elements and their contents as far as possible. Each <Menu> may optionally contain a <DefaultLayout> element which defines the default-layout for the current menu and all its sub-menus. If a menu has a <DefaultLayout> element then this will override any default-layout specified by a parent menu. The default-layout defines the suggested layout if a <Menu>

element does either not have <Layout> element or if it has an empty <Layout> element. For explanations of the various attributes see the <Menuname> element. If no default-layout has been specified then the layout as specified by the following elements should be assumed: <DefaultLayout show_empty="false" inline="false" inline_limit="4" inline_header="true" inline_alias="false"> <Merge type="menus"/><Merge type="files"/></DefaultLayout>

<Menuname [show_empty="..."] [inline="..."] [inline_limit="..."] [inline_header="..."] [inline_alias="..."]>

This element may only appear as a child of a <Layout> or <DefaultLayout> menu. Its contents references an immediate sub-menu of the current menu as defined with the <Menu> element, as such it should never contain a slash. If no such sub-menu exists the element should be ignored. This element may have various attributes, the default values are taken from the DefaultLayout key. The show_empty attribute defines whether a menu that contains no desktop entries and no sub-menus should be shown at all. The show_empty attribute can be "true" or "false". It may have an inline attribute that can be either "true" or "false". If the inline attribute is "true" the menu that is referenced may be copied into the current menu at the current point instead of being inserted as sub-menu of the current menu. The optional inline_limit attribute defines the maximum number of entries that can be inlined. If the sub-menu has more entries than inline_limit, the sub-menu will not be inlined. If the inline_limit is 0 (zero) there is no limit. The optional inline_header attribute defines whether an inlined menu should be preceded with a header entry listing the caption of the sub-menu. The inline_header attribute can be either "true" or "false". The optional inline_alias attribute defines whether a single inlined entry should adopt the caption of the inlined menu. In such case no additional header entry will be added regardless of the value of the inline_header attribute. The inline_alias attribute can be either "true" or "false". Example: if a menu has a sub-menu titled "WordProcessor" with a single entry "OpenOffice 4.2", and both inline="true" and inline_alias="true" are specified then this would result in the "OpenOffice 4.2" entry being inlined in the current menu but the "OpenOffice 4.2" caption of the entry would be replaced with "WordProcessor".

<Separator>

This element may only appear as a child of a <Layout> or <DefaultLayout> menu. It indicates a suggestion to draw a visual separator at this point in the menu. <Separator> elements at the start of a menu, at the end of a menu or that directly follow other <Separator> elements may be ignored.

<Merge type="menus"|"files"|"all"/>

This element may only appear as a child of a <Layout> or <DefaultLayout> menu. It indicates the point where desktop entries and sub-menus that are not explicitly mentioned within the <Layout> or <DefaultLayout> element are to be inserted. It has a type attribute that indicates which elements should be inserted: type="menus" means that all sub-menus that are not explicitly mentioned should be inserted in alphabetical order of their visual caption at this point. type="files" means that all desktop entries contained in this menu that are not explicitly mentioned should be inserted in alphabetical order of their visual

caption at this point. type="all" means that a mix of all sub-menus and all desktop entries that are not explicitly mentioned should be inserted in alphabetical order of their visual caption at this point. Each <Layout> or <DefaultLayout> element shall have exactly one <Merge type="all"> element or it shall have exactly one <Merge type="files"> and exactly one <Merge type="menus"> element. An exception is made for a completely empty <Layout> element which may be used to indicate that the default-layout should be used instead.

Merging

Sometimes two menu layouts need to be merged. This is done when folding in legacy menu hierarchies (see the section called “Legacy Menu Hierarchies”) and also for files specified in <MergeFile> elements. A common case is that per-user menu files might merge the system menu file. Merging is also used to avoid cut-and-paste, for example to include a common submenu in multiple menu files.

Merging involves a base <Menu> and a merged <Menu>. The base is the "target" menu and the merged <Menu> is being added to it. The result of the merge is termed the "combined menu."

As a preparatory step, the goal is to resolve all files into XML elements. To do so, traverse the entire menu tree. For each <MergeFile>, <MergeDir>, or <LegacyDir> element, replace the <MergeFile>, <MergeDir>, or <LegacyDir> element with the child elements of the root <Menu> of the file(s) being merged. As a special exception, remove the <Name> element from the root element of each file being merged. To generate a <Menu> based on a <LegacyDir>, see the section called “Legacy Menu Hierarchies”.

Continue processing until no <MergeFile>, <MergeDir>, or <LegacyDir> elements remain, taking care to avoid infinite loops caused by files that reference one another.

Once all files have been loaded into a single tree, scan the tree recursively performing these steps to remove duplicates:

1. Consolidate child menus. Each group of child <Menu>s with the same name must be consolidated into a single child menu with that name. Concatenate the child elements of all menus with the same name, in the order that they appear, and insert those elements as the children of the *last* menu with that name. Delete all the newly empty <Menu> elements, keeping the last one.

2. Expand <DefaultAppDirs> and <DefaultDirectoryDirs> elements to <AppDir> and <DirectoryDir> elements. Consolidate duplicate <AppDir>, <DirectoryDir>, and <Directory> elements by keeping the last one. For <Directory> elements that refer to distinct directory entries, all of them should be kept - if the last one points to a nonexistent file, the one before that can be used instead, and so forth.

3. Recurse into each child <Menu>, performing this list of steps for each child in order.

After recursing once to remove duplicates, recurse a second time to resolve <Move>

elements for each menu starting with any child menu before handling the more top level menus. So the deepest menus have their <Move> operations performed first. Within each <Menu>, execute <Move> operations in the order that they appear. If the destination path does not exist, simply relocate the origin <Menu> element, and change its <Name> field to match the destination path. If the origin path does not exist, do nothing. If both paths exist, take the origin <Menu> element, delete its <Name> element, and prepend its remaining child elements to the destination <Menu> element.

If any <Move> operations affect a menu, then re-run the steps to resolve duplicates in case any duplicates have been created.

Finally, for each <Menu> containing a <Deleted> element which is not followed by a <NotDeleted> element, remove that menu and all its child menus.

Merged menu elements are kept in order because <Include> and <Exclude> elements later in the file override <Include> and <Exclude> elements earlier in the file. This means that if the user's menu file merges the system menu file, the user can always override what the system menu specifies by placing elements after the <MergeFile> that incorporates the system file.

To prevent that a desktop entry from one party inadvertently cancels out the desktop entry from another party because both happen to get the same desktop-file id it is recommended that providers of desktop-files ensure that all desktop-file ids start with a vendor prefix. A vendor prefix consists of [a-zA-Z] and is terminated with a dash ("-"). Open Source projects and commercial parties are encouraged to use a word or phrase, preferably their name, as prefix for which they hold a trademark. Open Source applications can also ask to make use of the vendor prefix of another open source project (such as GNOME or KDE) they consider themselves affiliated with, at the discretion of these projects.

For example, to ensure that GNOME applications start with a vendor prefix of "gnome-", it could either add "gnome-" to all the desktop files it installs in *datadir*/applications/ or it could install desktop files in a *datadir*/applications/gnome subdirectory. When including legacy menu hierarchies the prefix argument of the <LegacyDir> element can be used to specify a prefix.

Generating the menus

After merging the menus, the result should be a single menu layout description. For each <Menu>, we have a list of directories where desktop entries can be found, a list of directories where directory entries can be found, and a series of <Include> and <Exclude> directives.

For each <Menu> element, build a pool of desktop entries by collecting entries found in each <AppDir> for the menu element. If two entries have the same desktop-file id, the entry for the earlier (closer to the top of the file) <AppDir> must be discarded. Next, add to the pool the entries for any <AppDir>s specified by ancestor <Menu> elements. If a parent menu has a duplicate entry (same desktop-file id), the entry for the child menu has priority.

Next, walk through all <Include> and <Exclude> statements. For each <Include>, match the rules against the pool of all desktop entries. For each desktop entry that matches one of the rules, add it to the menu to be displayed and mark it as having been allocated. For each <Exclude>, match the rules against the currently-included desktop entries. For each desktop entry that matches, remove it again from the menu. Note that an entry that is included in a menu but excluded again by a later <Exclude> is still considered allocated (for the purposes of <OnlyUnallocated>) even though that entry no longer appears in the menu.

Two passes are necessary, once for regular menus where any entry may be matched, and once for <OnlyUnallocated> menus where only entries which have not been marked as allocated may be matched.

The result is a tree of desktop entries, of course.

Legacy Menu Hierarchies

Traditionally, menus were defined as a filesystem hierarchy, with each filesystem directory corresponding to a submenu. Implementations of this specification must be able to load these old-style hierarchies as specified in this section.

The general approach is: the legacy hierarchy is converted into a <Menu>, and then this menu layout is merged with the menu that specified <LegacyDir>.

Desktop entries in the legacy hierarchy should be added to the pool of desktop entries as if the <LegacyDir> were an <AppDir>. Directory entries in the legacy hierarchy should be added to the pool of directory entries as if the <LegacyDir> were a <DirectoryDir>. This can be trivially implemented by adding appropriate <AppDir> and <DirectoryDir> statements to the root legacy <Menu>. There is one slight complexity, namely the "prefix" attribute of <LegacyDir>.

The menu layout corresponds conceptually to the following, though actually generating the XML is not necessary:

- For each directory in the legacy hierarchy, a <Menu> is created with the same <Name> as the directory on disk.
- This menu then contains an <Include> element that includes each desktop entry in the directory. That is, it should have a <Filename>*Foo/Bar/foo.desktop*</Filename> for each desktop entry in the directory.

  As a special exception, if a desktop entry in a directory contains a `Categories` field, that desktop entry should *not* be included in the legacy menu. That is, no <Include> element should be generated for the entry. This allows a desktop entry to be installed in a legacy location but still work optimally with the menu system specified in this document.
- If the legacy directory contains a ".directory" file, then a <Directory> element should be generated that points to said ".directory" file.
- Legacy desktop entries should not be assigned any `Categories` fields if they didn't have them already, except that all legacy entries should have the "Legacy"

category added to allow menu files to treat them specially. (If the same directory is given as both a <LegacyDir> and an <AppDir>, its desktop entries should be labeled "Legacy" only if the <LegacyDir> appears later in the file than the <AppDir>.)

For example, say we have the following legacy directory hierarchy:

```
/usr/share/applnk
    /usr/share/applnk/.directory
    /usr/share/applnk/bar.desktop
    /usr/share/applnk/System
        /usr/share/applnk/System/.directory
        /usr/share/applnk/System/foo.desktop
```

Conceptually that is converted to the following <Menu>:

```
<!DOCTYPE Menu PUBLIC "-//freedesktop//DTD Menu 1.0//EN"
"http://www.freedesktop.org/standards/menu-spec/menu-1.0.dtd">

<Menu>
  <Name>Applications</Name>
  <AppDir>/usr/share/applnk</AppDir>
  <DirectoryDir>/usr/share/applnk</DirectoryDir>
  <Directory>.directory</Directory>
  <Include>
    <Filename>bar.desktop</Filename>
  </Include>
  <Menu>
    <Name>System</Name>
    <AppDir>/usr/share/applnk/System</AppDir>
    <DirectoryDir>/usr/share/applnk/System</DirectoryDir>
    <Directory>.directory</Directory>
    <Include>
      <Filename>foo.desktop</Filename>
    </Include>
  </Menu>
</Menu>
```

This <Menu> is then merged as if it were in a file and loaded with <MergeFile>.

Example Menu File

```
<!DOCTYPE Menu PUBLIC "-//freedesktop//DTD Menu 1.0//EN"
"http://www.freedesktop.org/standards/menu-spec/menu-1.0.dtd">

<Menu>
  <Name>Applications</Name>
  <Directory>Applications.directory</Directory>

  <-- Search the default locations -->
  <DefaultAppDirs/>
  <DefaultDirectoryDirs/>

  <-- Merge third-party submenus -->
  <MergeDir>applications-merged</MergeDir>

  <-- Merge legacy hierarchy -->
  <LegacyDir>/usr/share/applnk</LegacyDir>
```

```
    <-- Define default layout -->
    <DefaultLayout>
      <Merge type="menus"/>
      <Merge type="files"/>
      <Separator/>
      <Menuname>More</Menuname>
    </DefaultLayout>

    <-- some random moves, maybe to clean up legacy dirs,
           maybe from menu editing -->
    <Move>
      <Old>Foo</Old>
      <New>Bar</New>
      <Old>Foo2</Old>
      <New>Bar2</New>
    </Move>

    <-- A preferences submenu, kept in a separate file
           so it can also be used standalone -->
    <Menu>
      <Name>Preferences</Name>
      <Directory>Preferences.directory</Directory>
      <MergeFile>preferences.menu</MergeFile>
    </Menu>

    <-- An Office submenu, specified inline -->
    <Menu>
      <Name>Office</Name>
      <Directory>Office.directory</Directory>
      <Include>
        <Category>Office</Category>
      </Include>
      <Exclude>
        <Filename>foo.desktop</Filename>
      </Exclude>
    </Menu>

  </Menu>
```

A. Registered Categories

This section contains a number of well known categories and suggestions on how to use them. The list of Main Categories consist of those categories that every conforming desktop environment MUST support. By including one of these categories in an application's desktop entry file the application will be ensured that it will show up in a section of the application menu dedicated to this category. The list of Additional Categories provides categories that can be used to provide more fine grained information about the application. Additional Categories should always be used in combination with one of the Main Categories.

The table below lists all Main Categories. Note that category names are case-sensitive.

Main Category	Description	Notes
AudioVideo	A multimedia (audio/video) application	

Main Category	Description	Notes
Audio	An audio application	Desktop entry must include AudioVideo as well
Video	A video application	Desktop entry must include AudioVideo as well
Development	An application for development	
Education	Educational software	
Game	A game	
Graphics	Graphical application	
Network	Network application such as a web browser	
Office	An office type application	
Settings	Settings applications	Entries may appear in a separate menu or as part of a "Control Center"
System	System application, "System Tools" such as say a log viewer or network monitor	
Utility	Small utility application, "Accessories"	

The table below describes Additional Categories. The Related Categories column lists one or more categories that are suggested to be used in conjunction with the Additional Category. Note that at least one Main Category must be included in the desktop entry's list of categories. If multiple Main Categories are included in a single desktop entry file, the entry may appear more than once in the menu. If the Related Categories column is blank, the Additional Category can be used with any Main Category.

Additional Category	Description	Related Categories
Building	A tool to build applications	Development
Debugger	A tool to debug applications	Development
IDE	IDE application	Development
GUIDesigner	A GUI designer application	Development
Profiling	A profiling tool	Development
RevisionControl	Applications like cvs or subversion	Development
Translation	A translation tool	Development

Additional Category	Description	Related Categories
Calendar	Calendar application	Office
ContactManagement	E.g. an address book	Office
Database	Application to manage a database	Office or Development or AudioVideo
Dictionary	A dictionary	Office;TextTools
Chart	Chart application	Office
Email	Email application	Office;Network
Finance	Application to manage your finance	Office
FlowChart	A flowchart application	Office
PDA	Tool to manage your PDA	Office
ProjectManagement	Project management application	Office;Development
Presentation	Presentation software	Office
Spreadsheet	A spreadsheet	Office
WordProcessor	A word processor	Office
2DGraphics	2D based graphical application	Graphics
VectorGraphics	Vector based graphical application	Graphics;2DGraphics
RasterGraphics	Raster based graphical application	Graphics;2DGraphics
3DGraphics	3D based graphical application	Graphics
Scanning	Tool to scan a file/text	Graphics
OCR	Optical character recognition application	Graphics;Scanning
Photography	Camera tools, etc.	Graphics or Office
Publishing	Desktop Publishing applications and Color Management tools	Graphics or Office
Viewer	Tool to view e.g. a graphic or pdf file	Graphics or Office
TextTools	A text tool utiliy	Utility
DesktopSettings	Configuration tool for the GUI	Settings
HardwareSettings	A tool to manage hardware components, like sound cards, video	Settings

Additional Category	Description	Related Categories
	cards or printers	
Printing	A tool to manage printers	HardwareSettings;Settings
PackageManager	A package manager application	Settings
Dialup	A dial-up program	Network
InstantMessaging	An instant messaging client	Network
Chat	A chat client	Network
IRCClient	An IRC client	Network
FileTransfer	Tools like FTP or P2P programs	Network
HamRadio	HAM radio software	Network or Audio
News	A news reader or a news ticker	Network
P2P	A P2P program	Network
RemoteAccess	A tool to remotely manage your PC	Network
Telephony	Telephony via PC	Network
TelephonyTools	Telephony tools, to dial a number, manage PBX, ...	Utility
VideoConference	Video Conference software	Network
WebBrowser	A web browser	Network
WebDevelopment	A tool for web developers	Network or Development
Midi	An app related to MIDI	AudioVideo;Audio
Mixer	Just a mixer	AudioVideo;Audio
Sequencer	A sequencer	AudioVideo;Audio
Tuner	A tuner	AudioVideo;Audio
TV	A TV application	AudioVideo;Video
AudioVideoEditing	Application to edit audio/video files	Audio or Video or AudioVideo
Player	Application to play audio/video files	Audio or Video or AudioVideo
Recorder	Application to record audio/video files	Audio or Video or AudioVideo
DiscBurning	Application to burn a disc	AudioVideo

Additional Category	Description	Related Categories
ActionGame	An action game	Game
AdventureGame	Adventure style game	Game
ArcadeGame	Arcade style game	Game
BoardGame	A board game	Game
BlocksGame	Falling blocks game	Game
CardGame	A card game	Game
KidsGame	A game for kids	Game
LogicGame	Logic games like puzzles, etc	Game
RolePlaying	A role playing game	Game
Simulation	A simulation game	Game
SportsGame	A sports game	Game
StrategyGame	A strategy game	Game
Art	Software to teach arts	Education
Construction		Education
Music	Musical software	AudioVideo;Education
Languages	Software to learn foreign languages	Education
Science	Scientific software	Education
ArtificialIntelligence	Artificial Intelligence software	Education;Science
Astronomy	Astronomy software	Education;Science
Biology	Biology software	Education;Science
Chemistry	Chemistry software	Education;Science
ComputerScience	ComputerSience software	Education;Science
DataVisualization	Data visualization software	Education;Science
Economy	Economy software	Education
Electricity	Electricity software	Education;Science
Geography	Geography software	Education
Geology	Geology software	Education;Science
Geoscience	Geoscience software	Education;Science
History	History software	Education
ImageProcessing	Image Processing software	Education;Science
Literature	Literature software	Education

Additional Category	Description	Related Categories
Math	Math software	Education;Science
NumericalAnalysis	Numerical analysis software	Education;Science;Math
MedicalSoftware	Medical software	Education;Science
Physics	Physics software	Education;Science
Robotics	Robotics software	Education;Science
Sports	Sports software	Education
ParallelComputing	Parallel computing software	Education;Science;ComputerScience
Amusement	A simple amusement	
Archiving	A tool to archive/backup data	Utility
Compression	A tool to manage compressed data/archives	Utility;Archiving
Electronics	Electronics software, e.g. a circuit designer	
Emulator	Emulator of another platform, such as a DOS emulator	System or Game
Engineering	Engineering software, e.g. CAD programs	
FileTools	A file tool utility	Utility or System
FileManager	A file manager	System;FileTools
TerminalEmulator	A terminal emulator application	System
Filesystem	A file system tool	System
Monitor	Monitor application/applet that monitors some resource or activity	System
Security	A security tool	Settings or System
Accessibility	Accessibility	Settings or Utility
Calculator	A calculator	Utility
Clock	A clock application/applet	Utility
TextEditor	A text editor	Utility
Documentation	Help or documentation	

Additional Category	Description	Related Categories
Core	Important application, core to the desktop such as a file manager or a help browser	
KDE	Application based on KDE libraries	QT
GNOME	Application based on GNOME libraries	GTK
GTK	Application based on GTK+ libraries	
Qt	Application based on Qt libraries	
Motif	Application based on Motif libraries	
Java	Application based on Java GUI libraries, such as AWT or Swing	
ConsoleOnly	Application that only works inside a terminal (text-based or command line application)	

The table below describes Reserved Categories. Reserved Categories have a specific desktop specific meaning that has not been standardized (yet). Desktop entry files that use a reserved category MUST also include an appropriate OnlyShowIn= entry to restrict themselves to those environments that properly support the reserved category as used.

Reserved Category	Description
Screensaver	A screen saver (launching this desktop entry should activate the screen saver)
TrayIcon	An application that is primarily an icon for the "system tray" or "notification area" (apps that open a normal window and just happen to have a tray icon as well should not list this category)
Applet	An applet that will run inside a panel or another such application, likely desktop specific
Shell	A shell (an actual specific shell such as `bash` or `tcsh`, not a TerminalEmulator)

B. Registered OnlyShowIn Environments

Remember, these are case-sensitive. "KDE" not "kde" should be used.

OnlyShowIn Value	Environment
GNOME	GNOME Desktop
KDE	KDE Desktop
ROX	ROX Desktop
XFCE	XFCE Desktop
Old	Legacy menu systems

C. Integrating your application in the menus

Adding menu items

The following steps describe how a third party application can add menu items to the menu system:

- Install desktop entries to *datadir*/applications/ for each menu item. Please namespace the filename, as in "vendor-foo.desktop", or use a subdirectory of *datadir*/applications/ so you have "vendor/foo.desktop." Please be sure all desktop entries are valid (see the desktop-file-utils package for a validation utility).
- Install an XML menu file to *sysconfdir*/desktop/menus/applications-merged/ to add any submenus, if your desktop entries aren't already included in some common categories.
- Install any directory entries needed for your submenus to *datadir*/desktop-directories/, taking care to namespace and validate the directory entries.

Install Locations

If an application is intended to be installed by root on a system wide basis then /usr/share is recommended to be used as value for *datadir* and /etc/xdg is recommended to be used as value for *sysconfdir*. In case the /usr/share hierarchy is not writable it is recommended to use /usr/local/share as value for *datadir* instead.

If an application is intended to be installed by an unprivileged user for exclusive use by that user only then `$XDG_DATA_HOME` should be used as value for *datadir* and `$XDG_CONFIG_HOME` should be used as value for *sysconfdir*. If `$XDG_DATA_HOME` is not set, the default value of $HOME/.local/share should be used for it. If `$XDG_CONFIG_HOME` is not set, the default value of $HOME/.config should be used for it.

Example

The company ShinyThings Inc. has developed an application named *WebMirror 1.0* and would like to add its own submenu to the system menus consisting of a *WebMirror* menu item and a *WebMirror Admin Tool* menu item. The company will use "shinythings" as its vendor id. For the purpose of this example all menu items will be available in two languages, English and Dutch. The language code for Dutch is *nl*.

First the company needs to create two .desktop files that describe the two menu items:

datadir/applications/shinythings-webmirror.desktop:

```
[Desktop Entry]
Encoding=UTF-8
Type=Application

Exec=webmirror
Icon=webmirror

Name=WebMirror
Name[nl]=WebSpiegel
```

and

datadir/applications/shinythings-webmirror-admin.desktop:

```
[Desktop Entry]
Encoding=UTF-8
Type=Application

Exec=webmirror-admintool
Icon=webmirror-admintool

Name=WebMirror Admin Tool
Name[nl]=WebSpiegel Administratie Tool
```

A .directory file needs to be installed to provide a title and icon for the sub-menu itself:

datadir/desktop-directories/shinythings-webmirror.directory:

```
[Desktop Entry]
Encoding=UTF-8

Icon=webmirror

Name=WebMirror
Name[nl]=WebSpiegel
```

And finally, a .menu file needs to be provided that links it all togther:

sysconfdir/menus/application-merged/shinythings-webmirror.menu:

```
<!DOCTYPE Menu PUBLIC "-//freedesktop//DTD Menu 1.0//EN"
"http://www.freedesktop.org/standards/menu-spec/menu-1.0.dtd">
<Menu>
  <Name>Applications</Name>
  <Menu>
    <Name>WebMirror</Name>
    <Directory>shinythings-webmirror.directory</Directory>
    <Include>
      <Filename>shinythings-webmirror.desktop</Filename>
      <Filename>shinythings-webmirror-admin.desktop</Filename>
    </Include>
  </Menu>
```

Backward Compatibility

For a limited time, installing a directory hierarchy to the old GNOME/KDE specific locations such as /usr/share/applnk and /usr/share/gnome/apps will continue to work as way to add your application to the menu system as well. There are two ways to support both the old and new menu systems at the same time:

- If you add a `Categories` line to the desktop entries in the legacy hierarchy, implementations of this specification will ignore their location in the legacy hierarchy, and arrange them according to `Categories` instead. This allows you to install a single desktop file that works in all cases, though on the down side it's in a legacy location.
- If you add the line `OnlyShowIn=Old;` to a desktop entry, then old legacy implementations that ignore `OnlyShowIn` will still show the desktop entry, but implementations of this specification will not. Thus you can add an "`OnlyShowIn=Old;`" entry to the legacy hierarchy, and a new-style desktop entry to *`datadir`*/applications/, and still get only one entry in the menus.

D. Implementation notes

Menu editing

To implement menu editing, the intent is that a per-user file is created. The per-user file should specify a <MergeFile> with the system wide file, so that system changes are inherited. When the user deletes a menu item, you add `<Exclude> <Filename>foo.desktop</Filename></Exclude>`. If the user adds a menu item, you use `<Include> <Filename>foo.desktop</Filename></Include>`.

If the user moves a folder you can use <Move> elements to represent the move. <Move> elements used for menu-editing should always be added to the most top-level menu to ensure that moves are performed in the order in which they are specified; moves specified in child menus are always performed before moves specified in a more top level menu regardless of their location in the menu file.

To delete a folder, simply append the <Deleted> element.

When adding a new folder or moving an existing folder, menu editing implementations are advised not to re-use the menu path of a previously deleted folder.

Menu editors probably need to do some kind of consolidation/compression to avoid an XML tree that grows infinitely over time.

Glossary

This glossary defines some of the terms used in this specification.

Desktop entry

A desktop entry is a file with a name ending in the ".desktop" extension which conforms to the desktop entry specification with `Type=Application`. It describes a menu item, including a name, an icon, and what to do when the item is selected. Desktop entries are also known as ".desktop files."

Desktop-File Id

The id to identify a desktop entry with. For example, if `/usr/share/applications` is specified as an <AppDir>, and `/opt/ude` as <LegacyDir prefix="foo-"> then `/usr/share/applications/foo/bar.desktop`, `/usr/share/applications/foo-bar.desktop` and `/opt/ude/Settings/bar.desktop` all have the same desktop-file id `foo-bar.desktop`

Directory entry

A directory entry is a file with a name ending in the ".directory" extension which conforms to the desktop entry specification with `Type=Directory`. It provides a localized name and an icon for a submenu. Directory entries are also known as ".directory files."

Menu path

A "menu path" is the path to a particular menu. Menu paths are always "relative" so never start with a slash character. The path to a menu is simply the <Name> of each parent of the menu, followed by the <Name> of the menu itself. For example, "Foo/Bar/Baz" is a valid menu path.

Relative path

The canonical path to a directory entry, relative to the <DirectoryDir> containing the entry. For example, if `/usr/share/desktop-directories` is specified as an <DirectoryDir>, the relative path to `/usr/share/desktop-directories/foo/bar.directory` is `foo/bar.directory`.