2020-04-27

Table of Contents

Introduction
File naming

- Desktop File ID

Basic format of the file

- Comments
- Group headers
- Entries

Possible value types
Localized values for keys
Recognized desktop entry keys
The `Exec` key
D-Bus Activation
Interfaces
Registering MIME Types
Additional applications actions

- Action identifier
- Action keys
- Implementation notes

Extending the format
A. Example Desktop Entry File
B. Currently reserved for use within KDE
C. Deprecated Items
D. The `Legacy-Mixed` Encoding (Deprecated)
E. Changes to this Specification

- Changes from version 1.4 to 1.5
- Changes from version 1.3 to 1.4
- Changes from version 1.2 to 1.3
- Changes from version 1.1 to 1.2
- Changes from version 1.0 to 1.1

Introduction

Both the KDE and GNOME desktop environments have adopted a similar format for "desktop entries", or configuration files describing how a particular program is to be launched, how it appears in menus, etc. It is to the larger community's benefit that a unified standard be agreed upon by all parties such that interoperation between the two environments, and indeed any additional environments that implement the specification, becomes simpler.

File naming

Desktop entry files should have the `.desktop` extension, except for files of `Type` `Directory` which should have the `.directory` extension. Determining file type on basis of extension makes determining the file type very easy

and quick. When no file extension is present, the desktop system should fall back to recognition via "magic detection".

For applications, the part of the name of the desktop file before the `.desktop` extension should be a valid D-Bus well-known name. This means that it is a sequence of non-empty elements separated by dots (U+002E FULL STOP), none of which starts with a digit, and each of which contains only characters from the set `[A-Za-z0-9_]`: ASCII letters, digits, dash (U+002D HYPHEN-MINUS) and underscore (U+005F LOW LINE).

The name of the desktop entry should follow the "reverse DNS" convention: it should start with a reversed DNS domain name controlled by the author of the application, in lower case. The domain name should be followed by the name of the application, which is conventionally written with words run together and initial capital letters (CamelCase). For example, if the owner of `example.org` writes "Foo Viewer", they might choose the name `org.example.FooViewer`, resulting in a file named `org.example.FooViewer.desktop`.

Well-known names containing the dash are allowed but not recommended, because the dash is not allowed in some related uses of reversed DNS names, such as D-Bus object paths and interface names, and Flatpak app IDs. If the author's domain name contains a dash, replacing it with an underscore is recommended: this cannot cause ambiguity, because underscores are not allowed in DNS domain names.

If the author's domain name contains a label starting with a digit, (which is not allowed in D-Bus well-known names), prepending an underscore to that element of the desktop entry name is recommended. For example, 7-zip.org might release an application named `org._7_zip.Archiver`.

Desktop File ID

Each desktop entry representing an application is identified by its *desktop file ID*, which is based on its filename.

To determine the ID of a desktop file, make its full path relative to the `$XDG_DATA_DIRS` component in which the desktop file is installed, remove the "applications/" prefix, and turn '/' into '-'.

For example `/usr/share/applications/foo/bar.desktop` has the desktop file ID `foo-bar.desktop`.

If multiple files have the same desktop file ID, the first one in the $XDG_DATA_DIRS precedence order is used.

For example, if `$XDG_DATA_DIRS` contains the default paths /usr/local/share:/usr/share, then `/usr/local/share/applications/org.foo.bar.desktop` and `/usr/share/applications/org.foo.bar.desktop` both have the same desktop file ID `org.foo.bar.desktop`, but only the first one will be used.

If both `foo-bar.desktop` and `foo/bar.desktop` exist, it is undefined which is selected.

If the desktop file is not installed in an `applications` subdirectory of one of the $XDG_DATA_DIRS components, it does not have an ID.

Basic format of the file

Desktop entry files are encoded in UTF-8. A file is interpreted as a series of lines that are separated by linefeed characters. Case is significant everywhere in the file.

Compliant implementations MUST not remove any fields from the file, even if they don't support them. Such fields must be maintained in a list somewhere, and if the file is "rewritten", they will be included. This ensures that any desktop-specific extensions will be preserved even if another system accesses and changes the file.

Comments

Lines beginning with a `#` and blank lines are considered comments and will be ignored, however they should be preserved across reads and writes of the desktop entry file.

Comment lines are uninterpreted and may contain any character (except for LF). However, using UTF-8 for comment lines that contain characters not in ASCII is encouraged.

Group headers

A group header with name `groupname` is a line in the format:

```
[groupname]
```

Group names may contain all ASCII characters except for `[` and `]` and control characters.

Multiple groups may not have the same name.

All `{key,value}` pairs following a group header until a new group header belong to the group.

The basic format of the desktop entry file requires that there be a group header named `Desktop Entry`. There may be other groups present in the file, but this is the most important group which explicitly needs to be supported. This group should also be used as the "magic key" for automatic MIME type detection. There should be nothing preceding this group in the desktop entry file but possibly one or more comments.

Entries

Entries in the file are `{key,value}` pairs in the format:

```
Key=Value
```

Space before and after the equals sign should be ignored; the `=` sign is the actual delimiter.

Only the characters `A-Za-z0-9-` may be used in key names.

As the case is significant, the keys `Name` and `NAME` are not equivalent.

Multiple keys in the same group may not have the same name. Keys in different groups may have the same name.

Possible value types

The value types recognized are `string`, `localestring`, `iconstring`, `boolean`, and `numeric`.

- Values of type `string` may contain all ASCII characters except for control characters.
- Values of type `localestring` are user displayable, and are encoded in UTF-8.
- Values of type `iconstring` are the names of icons; these may be absolute paths, or symbolic names for icons located using the algorithm described in the Icon Theme Specification. Such values are not user-displayable, and are encoded in UTF-8.
- Values of type `boolean` must either be the string `true` or `false`.

- Values of type `numeric` must be a valid floating point number as recognized by the `%f` specifier for `scanf` in the `C` locale.

The escape sequences `\s`, `\n`, `\t`, `\r`, and `\\` are supported for values of type `string`, `localestring` and `iconstring`, meaning ASCII space, newline, tab, carriage return, and backslash, respectively.

Some keys can have multiple values. In such a case, the value of the key is specified as a plural: for example, `string(s)`. The multiple values should be separated by a semicolon and the value of the key may be optionally terminated by a semicolon. Trailing empty strings must always be terminated with a semicolon. Semicolons in these values need to be escaped using `\;`.

Localized values for keys

Keys with type `localestring` and `iconstring` may be postfixed by [*LOCALE*], where *LOCALE* is the locale type of the entry. *LOCALE* must be of the form *lang_COUNTRY.ENCODING@MODIFIER*, where *_COUNTRY*, *.ENCODING*, and *@MODIFIER* may be omitted. If a postfixed key occurs, the same key must be also present without the postfix.

When reading in the desktop entry file, the value of the key is selected by matching the current POSIX locale for the `LC_MESSAGES` category against the *LOCALE* postfixes of all occurrences of the key, with the *.ENCODING* part stripped.

The matching is done as follows. If `LC_MESSAGES` is of the form *lang_COUNTRY.ENCODING@MODIFIER*, then it will match a key of the form *lang_COUNTRY@MODIFIER*. If such a key does not exist, it will attempt to match *lang_COUNTRY* followed by *lang@MODIFIER*. Then, a match against *lang* by itself will be attempted. Finally, if no matching key is found the required key without a locale specified is used. The encoding from the `LC_MESSAGES` value is ignored when matching.

If `LC_MESSAGES` does not have a *MODIFIER* field, then no key with a modifier will be matched. Similarly, if `LC_MESSAGES` does not have a *COUNTRY* field, then no key with a country specified will be matched. If `LC_MESSAGES` just has a *lang* field, then it will do a straight match to a key with a similar value. The following table lists possible matches of various `LC_MESSAGES` values in the order in which they are matched. Note that the *ENCODING* field isn't shown.

Table 1. Locale Matching

LC_MESSAGES value	Possible keys in order of matching
lang_COUNTRY@MODIFIER	*lang_COUNTRY@MODIFIER*, *lang_COUNTRY*, *lang@MODIFIER*, *lang*, default value
lang_COUNTRY	*lang_COUNTRY*, *lang*, default value
lang@MODIFIER	*lang@MODIFIER*, *lang*, default value
lang	*lang*, default value

For example, if the current value of the `LC_MESSAGES` category is `sr_YU@Latn` and the desktop file includes:

```
Name=Foo
Name[sr_YU]=...
Name[sr@Latn]=...
Name[sr]=...
```

then the value of the `Name` keyed by `sr_YU` is used.

Although icon names of type `iconstring` are localizable, they are not human-readable strings, so should typically not be handled by translation tools. Most applications are not expected to localize their icons;

exceptions might include icons containing text or culture-specific symbology.

Recognized desktop entry keys

Keys are either OPTIONAL or REQUIRED. If a key is OPTIONAL it may or may not be present in the file. However, if it isn't, the implementation of the standard should not blow up, it must provide some sane defaults.

Some keys only make sense in the context when another particular key is also present and set to a specific value. Those keys should not be used if the particular key is not present or not set to the specific value. For example, the `Terminal` key can only be used when the value of the `Type` key is `Application`.

If a REQUIRED key is only valid in the context of another key set to a specific value, then it has to be present only if the other key is set to the specific value. For example, the `URL` key has to be present when and only when when the value of the `Type` key is `Link`.

Some example keys: `Name[C]`, `Comment[it]`.

Table 2. Standard Keys

Key	Description	Value Type	REQ?	Type
`Type`	This specification defines 3 types of desktop entries: `Application` (type 1), `Link` (type 2) and `Directory` (type 3). To allow the addition of new types in the future, implementations should ignore desktop entries with an unknown type.	string	YES	
`Version`	Version of the Desktop Entry Specification that the desktop entry conforms with. Entries that confirm with this version of the specification should use `1.5`. Note that the version field is not required to be present.	string	NO	1-3
`Name`	Specific name of the application, for example "Mozilla".	localestring	YES	1-3
`GenericName`	Generic name of the application, for example "Web Browser".	localestring	NO	1-3
`NoDisplay`	`NoDisplay` means "this application exists, but don't display it in the menus". This can be useful to e.g. associate this application with MIME types, so that it gets launched from a file manager (or other apps), without having a menu entry for it (there are tons of good reasons for this, including e.g. the `netscape -remote`, or `kfmclient openURL` kind of stuff).	boolean	NO	1-3
`Comment`	Tooltip for the entry, for example "View sites on the Internet". The value should not be redundant with the values of `Name` and `GenericName`.	localestring	NO	1-3
`Icon`	Icon to display in file manager, menus, etc. If the name is an absolute path, the given file will be used. If the name is not an absolute path, the algorithm described in the Icon Theme Specification will be used to locate the icon.	iconstring	NO	1-3
`Hidden`	`Hidden` should have been called `Deleted`. It means the user deleted (at their level) something that was present (at an upper level, e.g. in the system dirs). It's strictly equivalent to the `.desktop` file not existing at all, as far	boolean	NO	1-3

Key	Description	Value Type	REQ?	Type
	as that user is concerned. This can also be used to "uninstall" existing files (e.g. due to a renaming) - by letting `make install` install a file with `Hidden=true` in it.			
`OnlyShowIn`, `NotShowIn`	A list of strings identifying the desktop environments that should display/not display a given desktop entry. By default, a desktop file should be shown, unless an OnlyShowIn key is present, in which case, the default is for the file not to be shown. If `$XDG_CURRENT_DESKTOP` is set then it contains a colon-separated list of strings. In order, each string is considered. If a matching entry is found in `OnlyShowIn` then the desktop file is shown. If an entry is found in `NotShowIn` then the desktop file is not shown. If none of the strings match then the default action is taken (as above). `$XDG_CURRENT_DESKTOP` should have been set by the login manager, according to the value of the `DesktopNames` found in the session file. The entry in the session file has multiple values separated in the usual way: with a semicolon. The same desktop name may not appear in both `OnlyShowIn` and `NotShowIn` of a group.	string(s)	NO	1-3
`DBusActivatable`	A boolean value specifying if D-Bus activation is supported for this application. If this key is missing, the default value is `false`. If the value is `true` then implementations should ignore the `Exec` key and send a D-Bus message to launch the application. See D-Bus Activation for more information on how this works. Applications should still include Exec= lines in their desktop files for compatibility with implementations that do not understand the DBusActivatable key.	boolean	NO	
`TryExec`	Path to an executable file on disk used to determine if the program is actually installed. If the path is not an absolute path, the file is looked up in the $PATH environment variable. If the file is not present or if it is not executable, the entry may be ignored (not be used in menus, for example).	string	NO	1
`Exec`	Program to execute, possibly with arguments. See the Exec key for details on how this key works. The `Exec` key is required if `DBusActivatable` is not set to `true`. Even if `DBusActivatable` is `true`, `Exec` should be specified for compatibility with implementations that do not understand `DBusActivatable`.	string	NO	1
`Path`	If entry is of type `Application`, the working directory to run the program in.	string	NO	1
`Terminal`	Whether the program runs in a terminal window.	boolean	NO	1

Key	Description	Value Type	REQ?	Type
`Actions`	Identifiers for application actions. This can be used to tell the application to make a specific action, different from the default behavior. The Application actions section describes how actions work.	string(s)	NO	1
`MimeType`	The MIME type(s) supported by this application.	string(s)	NO	1
`Categories`	Categories in which the entry should be shown in a menu (for possible values see the Desktop Menu Specification).	string(s)	NO	1
`Implements`	A list of interfaces that this application implements. By default, a desktop file implements no interfaces. See Interfaces for more information on how this works.	string(s)	NO	
`Keywords`	A list of strings which may be used in addition to other metadata to describe this entry. This can be useful e.g. to facilitate searching through entries. The values are not meant for display, and should not be redundant with the values of `Name` or `GenericName`.	localestring(s)	NO	1
`StartupNotify`	If true, it is KNOWN that the application will send a "remove" message when started with the DESKTOP_STARTUP_ID environment variable set. If false, it is KNOWN that the application does not work with startup notification at all (does not shown any window, breaks even when using StartupWMClass, etc.). If absent, a reasonable handling is up to implementations (assuming false, using StartupWMClass, etc.). (See the Startup Notification Protocol Specification for more details).	boolean	NO	1
`StartupWMClass`	If specified, it is known that the application will map at least one window with the given string as its WM class or WM name hint (see the Startup Notification Protocol Specification for more details).	string	NO	1
`URL`	If entry is Link type, the URL to access.	string	YES	2
`PrefersNonDefaultGPU`	If true, the application prefers to be run on a more powerful discrete GPU if available, which we describe as “a GPU other than the default one” in this spec to avoid the need to define what a discrete GPU is and in which cases it might be considered more powerful than the default GPU. This key is only a hint and support might not be present depending on the implementation.	boolean	NO	1
`SingleMainWindow`	If true, the application has a single main window, and does not support having an additional one opened. This key is used to signal to the implementation to avoid offering a UI to launch another window of the app. This key is only a hint and support might not be present depending on the implementation.	boolean	NO	1

The `Exec` key

The `Exec` key must contain a command line. A command line consists of an executable program optionally followed by one or more arguments. The executable program can either be specified with its full path or with the name of the executable only. If no full path is provided the executable is looked up in the $PATH environment variable used by the desktop environment. The name or path of the executable program may not contain the equal sign ("="). Arguments are separated by a space.

Arguments may be quoted in whole. If an argument contains a reserved character the argument must be quoted. The rules for quoting of arguments is also applicable to the executable name or path of the executable program as provided.

Quoting must be done by enclosing the argument between double quotes and escaping the double quote character, backtick character ("`"), dollar sign ("$") and backslash character ("\") by preceding it with an additional backslash character. Implementations must undo quoting before expanding field codes and before passing the argument to the executable program. Reserved characters are space (" "), tab, newline, double quote, single quote ("'"), backslash character ("\"), greater-than sign (">"), less-than sign ("<"), tilde ("~"), vertical bar ("|"), ampersand ("&"), semicolon (";"), dollar sign ("$"), asterisk ("*"), question mark ("?"), hash mark ("#"), parenthesis ("(") and (")") and backtick character ("`").

Note that the general escape rule for values of type string states that the backslash character can be escaped as ("\\") as well and that this escape rule is applied before the quoting rule. As such, to unambiguously represent a literal backslash character in a quoted argument in a desktop entry file requires the use of four successive backslash characters ("\\\\"). Likewise, a literal dollar sign in a quoted argument in a desktop entry file is unambiguously represented with ("\\$").

A number of special field codes have been defined which will be expanded by the file manager or program launcher when encountered in the command line. Field codes consist of the percentage character ("%") followed by an alpha character. Literal percentage characters must be escaped as `%%`. Deprecated field codes should be removed from the command line and ignored. Field codes are expanded only once, the string that is used to replace the field code should not be checked for field codes itself.

Command lines that contain a field code that is not listed in this specification are invalid and must not be processed, in particular implementations may not introduce support for field codes not listed in this specification. Extensions, if any, should be introduced by means of a new key.

Implementations must take care not to expand field codes into multiple arguments unless explicitly instructed by this specification. This means that name fields, filenames and other replacements that can contain spaces must be passed as a single argument to the executable program after expansion.

Although the `Exec` key is defined to have a value of the type string, which is limited to ASCII characters, field code expansion may introduce non-ASCII characters in arguments. Implementations must take care that all characters in arguments passed to the executable program are properly encoded according to the applicable locale setting.

Recognized field codes are as follows:

Code	Description
`%f`	A single file name (including the path), even if multiple files are selected. The system reading the desktop entry should recognize that the program in question cannot handle multiple file arguments, and it should should probably spawn and execute multiple copies of a program for each selected file if the program is not able to handle additional file arguments. If files are not on the local file system (i.e. are on HTTP or FTP locations), the files will be copied to the local file system and `%f` will be expanded to point at the temporary file. Used for programs that do not understand the URL syntax.
`%F`	A list of files. Use for apps that can open several local files at once. Each file is passed as a separate argument to the executable program.

Code	Description
%u	A single URL. Local files may either be passed as file: URLs or as file path.
%U	A list of URLs. Each URL is passed as a separate argument to the executable program. Local files may either be passed as file: URLs or as file path.
%d	Deprecated.
%D	Deprecated.
%n	Deprecated.
%N	Deprecated.
%i	The Icon key of the desktop entry expanded as two arguments, first --icon and then the value of the Icon key. Should not expand to any arguments if the Icon key is empty or missing.
%c	The translated name of the application as listed in the appropriate Name key in the desktop entry.
%k	The location of the desktop file as either a URI (if for example gotten from the vfolder system) or a local filename or empty if no location is known.
%v	Deprecated.
%m	Deprecated.

A command line may contain at most one %f, %u, %F or %U field code. If the application should not open any file the %f, %u, %F and %U field codes must be removed from the command line and ignored.

Field codes must not be used inside a quoted argument, the result of field code expansion inside a quoted argument is undefined. The %F and %U field codes may only be used as an argument on their own.

D-Bus Activation

Applications that support being launched by D-Bus must implement the following interface (given in D-Bus introspection XML format):

```
<interface name='org.freedesktop.Application'>
  <method name='Activate'>
    <arg type='a{sv}' name='platform_data' direction='in'/>
  </method>
  <method name='Open'>
    <arg type='as' name='uris' direction='in'/>
    <arg type='a{sv}' name='platform_data' direction='in'/>
  </method>
  <method name='ActivateAction'>
    <arg type='s' name='action_name' direction='in'/>
    <arg type='av' name='parameter' direction='in'/>
    <arg type='a{sv}' name='platform_data' direction='in'/>
  </method>
</interface>
```

The application must name its desktop file in accordance with the naming recommendations in the introduction section (e.g. the filename must be like `org.example.FooViewer.desktop`). The application must have a D-Bus service activatable at the well-known name that is equal to the desktop file name with the `.desktop` portion removed (for our example, `org.example.FooViewer`). The above interface must be implemented at an object path determined as follows: starting with the well-known D-Bus name of the application, change all dots to slashes and prefix a slash. If a dash ('-') is found, convert it to an underscore ('_'). For our example, this is `/org/example/FooViewer`.

The `Activate` method is called when the application is started without files to open.

The `Open` method is called when the application is started with files. The array of strings is an array of URIs, in UTF-8.

The `ActivateAction` method is called when Desktop Actions are activated. The `action-name` parameter is the name of the action.

All methods take a `platform-data` argument that is used in a similar way to how environment variables might be used. Current fields described by the specification are:

- `desktop-startup-id`: This should be a string of the same value as would be stored in the `DESKTOP_STARTUP_ID` environment variable, as specified by the Startup Notification Protocol Specification.
- `activation-token`: This should be a string of the same value as would be stored in the `XDG_ACTIVATION_TOKEN` environment variable, as specified by the XDG Activation protocol for Wayland.

Interfaces

The `Implements` key can be used to declare one or more interfaces that a desktop file implements.

Each interface name must follow the rules used for D-Bus interface names, but other than that, they have no particular meaning. For instance, listing an interface here does not necessarily mean that this application implements that D-Bus interface or even that such a D-Bus interface exists. It is entirely up to the entity who defined a particular interface to define what it means to implement it.

Although it is entirely up to the designer of the interface to decide what a given interface name means, here are some recommended "best practices":

- interfaces should require that application is DBusActivatable, including the requirement that the application's desktop file is named using the D-Bus "reverse DNS" convention
- the interface name should correspond to a D-Bus interface that the application exports on the same object path as it exports the org.freedesktop.Application interface
- if the interface wishes to allow for details about the implementation, it should do so by specifying that implementers add a group in their desktop file with the same name as the interface (eg: "[org.freedesktop.ImageAcquire]")

Recommendations notwithstanding, interfaces could specify almost any imaginable requirement including such (ridiculous) things as "when launched via the Exec line, the application is expected to present a window with the _FOO_IDENTIFIER property set, at which point an X client message will be sent to that window". Another example is "all implementations of this interface are expected to be marked NoDisplay and, on launch, will present no windows and will delete all of the user's files without confirmation".

Interface definers should take care to keep issues of backward and forward compatibility in mind when designing their interfaces.

Registering MIME Types

The `MimeType` key is used to indicate the MIME Types that an application knows how to handle. It is expected that for some applications this list could become long. An application is expected to be able to reasonably open files of these types using the command listed in the `Exec` key.

There should be no priority for MIME Types in this field, or any form of priority in the desktop file. Priority for applications is handled external to the `.desktop` files.

Additional applications actions

Desktop entries of type Application can include one or more actions. An action represents an additional way to invoke the application. Application launchers should expose them to the user (for example, as a submenu) within the context of the application. This is used to build so called "Quicklists" or "Jumplists".

Action identifier

Each action is identified by a string, following the same format as key names (see the section called “Entries”). Each identifier is associated with an action group that must be present in the `.desktop` file. The action group is a group named `Desktop Action %s`, where `%s` is the action identifier.

It is not valid to have an action group for an action identifier not mentioned in the `Actions` key. Such an action group must be ignored by implementors.

Action keys

The following keys are supported within each action group. If a REQUIRED key is not present in an action group, then the implementor should ignore this action.

Table 3. Action Specific Keys

Key	Description	Value Type	REQ?	
`Name`	Label that will be shown to the user. Since actions are always shown in the context of a specific application (that is, as a submenu of a launcher), this only needs to be unambiguous within one application and should not include the application name.	localestring	YES	
`Icon`	Icon to be shown together with the action. If the name is an absolute path, the given file will be used. If the name is not an absolute path, the algorithm described in the Icon Theme Specification will be used to locate the icon. Implementations may choose to ignore it.	iconstring	NO	
`Exec`	Program to execute for this action, possibly with arguments. See the Exec key for details on how this key works. The `Exec` key is required if `DBusActivatable` is not set to `true` in the main desktop entry group. Even if `DBusActivatable` is `true`, `Exec` should be specified for compatibility with implementations that do not understand `DBusActivatable`.	string	NO	

Implementation notes

Application actions should be supported by implementors. However, in case they are not supported, implementors can simply ignore the `Actions` key and the associated `Desktop Action` action groups, and keep using the `Desktop Entry` group: the primary way to describe and invoke the application is through the Name, Icon and Exec keys from the `Desktop Entry` group.

It is not expected that other desktop components showing application lists (software installers, for instance) will provide any user interface for these actions. Therefore applications must only include actions that make sense as general launchers.

Extending the format

If the standard is to be amended with a new `{key,value}` pair which should be applicable to all supporting parties, a group discussion will take place. This is the preferred method for introducing changes. If one particular party wishes to add a field for personal use, they should prefix the key with the string `X-`*`PRODUCT`*, e.g. `X-NewDesktop-Foo`, following the precedent set by other IETF and RFC standards.

Alternatively, fields can be placed in their own group, where they may then have arbitrary key names. If this is the case, the group should follow the scheme outlined above, i.e. `[X-`*`PRODUCT GROUPNAME`*`]` or something similar. These steps will avoid namespace clashes between different yet similar environments.

A. Example Desktop Entry File

```
[Desktop Entry]
Version=1.0
Type=Application
Name=Foo Viewer
Comment=The best viewer for Foo objects available!
TryExec=fooview
Exec=fooview %F
Icon=fooview
MimeType=image/x-foo;
Actions=Gallery;Create;

[Desktop Action Gallery]
Exec=fooview --gallery
Name=Browse Gallery

[Desktop Action Create]
Exec=fooview --create-new
Name=Create a new Foo!
Icon=fooview-new
```

B. Currently reserved for use within KDE

For historical reasons KDE is using some KDE-specific extensions that are currently not prefixed by a `X-KDE-` prefix.

- KDE specific keys: `ServiceTypes`, `DocPath`, `InitialPreference`
- KDE specific types: `ServiceType`, `Service` and `FSDevice`

KDE also used the `Keywords` key before it was standardized, using commas instead of semi-colons as separators. At the time of standardization, the field had been prefixed with a `X-KDE` prefix, but the Trinity fork still used the non-prefixed variant.

KDE uses the following additional keys for desktop entries of the `FSDevice` type.

Table B.1. FSDevice Specific Keys

Key	Description	Value Type
`Dev`	The device to mount.	string
`FSType`	The type of file system to try to mount.	string

Key	Description	Value Type
`MountPoint`	The mount point of the device in question.	string
`ReadOnly`	Specifies whether or not the device is read only.	boolean
`UnmountIcon`	Icon to display when device is not mounted. Mounted devices display icon from the `Icon` key.	iconstring

C. Deprecated Items

As this standard is quite old there are some deprecated items that may or may not be used by several implementations.

- `Type=MimeType` is deprecated as there is a new standard for this now, see the Shared MIME-info Database specification for more information. In consequence the Keys `Patterns` (various file name extensions associated with the MIME type) and `DefaultApp` (the default application associated with this MIME type) are also deprecated.
- Using `.kdelnk` instead of `.desktop` as the file extension is deprecated.
- Using `[KDE Desktop Entry]` instead of `[Desktop Entry]` as header is deprecated.
- The `Encoding` key is deprecated. It was used to specify whether keys of type `localestring` were encoded in UTF-8 or in the specified locale. Possible values are `UTF-8` and `Legacy-Mixed`. See Appendix D, *The `Legacy-Mixed` Encoding (Deprecated)* for more details.
- Deprecated `Exec` field codes: `%m` (the mini-icon associated with the desktop entry, this should be expanded as two arguments, `--miniicon` and the content of the `MiniIcon` key, it can also be ignored by expanding it to no arguments), %v (the device as listed in the `Dev` key in the desktop file), %d (the directory of a file), %D (the directories of files), %n (the base name of a file) and %N (the base names of files).
- Deprecated keys: `MiniIcon` (small icon for menus, etc.), `TerminalOptions` (if the program runs in a terminal, any options that should be passed to the terminal emulator before actually executing the program), `Protocols`, `Extensions`, `BinaryPattern`, `MapNotify`.
- The `SwallowTitle` and `SwallowExec` keys are deprecated. The `SwallowTitle` key is of type `localestring` and specifies the title of the window if is swallowed onto the panel. The `SwallowExec` key is of type `string` and specifies the program to exec if swallowed app is clicked.
- The `SortOrder` key is deprecated. It is of type `string(s)` and may be used to specify the order in which to display files. The Desktop Menu Specification defines another mechanism for defining the order of menu items.
- The `FilePattern` key is deprecated. The value is a list of regular expressions to match against for a file manager to determine if this entry's icon should be displayed. Usually simply the name of the main executable and friends.
- Historically some booleans have been represented by the numeric entries `0` or `1`. With this version of the standard they are now to be represented as a boolean string. However, if an implementation is reading a pre-1.0 desktop entry, it should interpret `0` and `1` as `false` and `true`, respectively.
- Historically lists have been comma separated. This is inconsistent with other lists which are separated by a semicolon. When reading a pre-1.0 desktop entry, comma separated lists should continue to be supported.

D. The `Legacy-Mixed` Encoding (Deprecated)

The `Legacy-Mixed` encoding corresponds to the traditional encoding of desktop files in older versions of the GNOME and KDE desktop files. In this encoding, the encoding of each `localestring` key is determined by the locale tag for that key, if any, instead of being UTF-8. For keys without a locale tag, the value must contain only ASCII characters.

If the file specifies an unsupported encoding, the implementation should either ignore the file, or, if the user has requested a direct operation on the file (such as opening it for editing), display an appropriate error indication to the user.

In the absence of an `Encoding` key, the implementation may choose to autodetect the encoding of the file by using such factors as:

- The location of the file on the file system
- Whether the contents of the file are valid UTF-8

If the implementation does not perform such auto-detection, it should treat a file without an `Encoding` key in the same way as a file with an unsupported `Encoding` key.

If the locale tag includes an *`.ENCODING`* part, then that determines the encoding for the line. Otherwise, the encoding is determined by the language, or *`lang_COUNTRY`* pair from the locale tag, according to the following table.

Encoding	Aliases	Tags
ARMSCII-8 (*)		hy
BIG5		zh_TW
CP1251		be bg
EUC-CN	GB2312	zh_CN
EUC-JP		ja
EUC-KR		ko
GEORGIAN-ACADEMY (*)		
GEORGIAN-PS (*)		ka
ISO-8859-1		br ca da de en es eu fi fr gl it nl no pt sv wa
ISO-8859-2		cs hr hu pl ro sk sl sq sr
ISO-8859-3		eo
ISO-8859-5		mk sp
ISO-8859-7		el
ISO-8859-9		tr
ISO-8859-13		lt lv mi
ISO-8859-14		cy ga
ISO-8859-15		et
KOI8-R		ru
KOI8-U		uk
TCVN-5712 (*)	TCVN	vi
TIS-620		th

Encoding	Aliases	Tags
VISCII		

Encoding

The name given here is listed here is typically the canonical name for the encoding in the GNU C Library's `iconv` facility. Encodings marked with (*) are not currently supported by the GNU C Library; for this reason, implementations may choose to ignore lines in desktop files that resolve to this encoding. Desktop files with these encodings are currently rare or non-existent.

Aliases

Other names for the encoding found in existing desktop files.

Tags

Language tags for which this is the default encoding.

This table above covers all tags and encodings that are known to be currently in use. Implementors may choose to support encodings not in the above set. For tags without defaults listed in the above table, desktop file creators must specify the *`.ENCODING`* part of the locale tag.

Matching the *`.ENCODING`* part of the locale tag against a locale name or alias should be done by stripping all punctuation characters from both the tag and the name or alias, converting both name and alias to lowercase, and comparing the result. This is necessary because, for example, `Big5` is frequently found instead of `BIG5` and `georgianacademy` instead of `GEORGIAN-ACADEMY`. Desktop files creators should, however, use the name as it appears in the "Encoding" column above.

E. Changes to this Specification

Changes from version 1.4 to 1.5

Add SingleMainWindow key.

Changes from version 1.3 to 1.4

Add PrefersNonDefaultGPU key.

Changes from version 1.2 to 1.3

New type "iconstring", similar to "localestring" but not user-displayable.

Changes from version 1.1 to 1.2

New section "File naming", to formalize file names, with relation to D-Bus naming.

Add Implements key.

Specify XDG_CURRENT_DESKTOP.