APERTIS

Application entry points

Contents

Requirements 2
Security and privacy considerations 3
Menu entries 4
Agents 4
Non-requirements 4
Recommendation 4
App identification 4
Desktop entries 5
Simple applications (one entry point) 9
Services 10
Entry points which do not appear in the menus 10
Multiple-view applications 10
Appendix: GSettingsSchema-based entry point registration prior to October 2015 12
Appendix: other approaches to multiple-view applications 14
One `Desktop Action` per view 15
One Apertis-specific menu entry per view 16

Requirements

Application bundles[1] may contain *application entry points*, which are any of these things:

- a graphical program[2] that would normally appear in a menu
- a graphical program that would not normally appear in a menu, but can be launched in some other way, for example as a content-type handler[3]
- a user service[4] that starts during device startup
- a user service that is started on-demand

Desktop environments provide metadata about these programs so that they can be launched.

At least the following use-cases exist:

- mildenhall-launcher displays a categorized menu of user-facing programs. Typical graphical programs such as the Rhayader web browser must appear here, with a name and an icon.
- It must be possible to translate the name into multiple languages, with a default (international English) name used for languages where there is no specific translation.
- Different manufacturers' launcher implementations might have a different taxonomy of categories for programs.

[1] https://martyn.pages.apertis.org/apertis-website/glossary/#application-bundles
[2] https://martyn.pages.apertis.org/apertis-website/glossary/#graphical-program
[3] https://martyn.pages.apertis.org/apertis-website/concepts/content_hand-over/
[4] https://martyn.pages.apertis.org/apertis-website/glossary/#user-service

- If two graphical programs have the same user-facing name, it might be useful to be able to provide a longer distinguishing name. For example, if both Chrome and Firefox are installed, they might be called “Firefox Browser” and “Chrome Browser”, but if only one is installed, it might simply be called “Browser”.
- Certain graphical programs should be hidden from the menu, but treated as a first-class program during user interaction. As of October 2015, the Canterbury application manager has hard-coded special cases for various removable storage browsing applications; an improved metadata format would allow these special cases to be generalized.
- Some graphical programs present multiple *views* which may appear separately in menus, but are all implemented in terms of the same running process. For example, the Frampton audio player appears in the menu three times, as “Albums”, “Artists” and “Songs”. However, ideally there would only be one Frampton HMI process at any given time, even if the user switches between views.
- Some programs should be started during device startup or user login.
- In the SDK images, Apertis applications and services should not necessarily be listed in the XFCE menus, and XFCE applications should not be listed in the “device simulator”.

Security and privacy considerations

The list of installed store application bundles in /Applications is considered to be private information, for several reasons:

- the general operating principle for Apertis’ app framework is that apps must not affect each other, except where given permission to interact, ensuring “loose coupling” between apps
- the presence of certain app bundles might be considered to be sensitive (for example, app bundles correlated with political or religious views)
- the complete list could be used for user fingerprinting, for example guessing that users of an online service share a device by observing that they have the same list of app-bundles

The list of installed entry-points is almost equivalent to the list of store application bundles and has similar considerations. However, some components cannot work without a list of store application bundles, or a list of their entry points. This leads to some privacy requirements:

- Certain platform components such as the Canterbury app manager, the Didcot content handover service, and the mildenhall-launcher app-launching HMI require the ability to list store application bundles and/or their entry points. They must be able to do so.
- Store applications with special permissions might also be allowed to list store application bundles and/or their entry points.
- Store applications may list the entry points that advertise a particular

public interface, as defined in the Interface discovery[5] design.
- Store applications without special permissions must not be able to enumerate store application bundles that do not contain an entry point advertising a public interface, either directly or by enumerating entry points and inferring the existence of a bundle from its entry points.

Unlike store application bundles, we suggest that the list of installed built-in application bundles in `/usr/Applications` should *not* be considered to be private. This list will be the same for every instance of the same platform image, so an application author could learn this list by querying the platform image variant and version, then matching that to a pre-prepared list of application bundles known to exist in their own copy of the same image. Conversely, because this list is the same for every instance of the same platform image, it is not useful for user fingerprinting.

Menu entries

Optionally, a single entry point may be specified to provide an icon for presentation in the application launcher. If no icon is presented it won't be obvious to the user that they have the application installed, so the application store screening process should carefully consider whether an application should be allowed to install services and type handlers with no icon for the launcher.

The Applications concept design has historically assumed that application bundles should be constrained to contain at most one menu entry. However, one of the reference app-bundles developed as part of Apertis (the Frampton media player[6]) has multiple menu entries, so this document has assumed that this constraint is no longer desired.

Agents

Agents should be specified as entry points, with a localized list of names for the agent, along with the location of the executable file to launch. Since agents can be long running and have an impact on device performance, any application with an agent should also set the `agent` permission[7] so the user can choose not to install the application.

Non-requirements

System services[8] are outside the scope of this design.

[5] https://martyn.pages.apertis.org/apertis-website/concepts/interface_discovery/
[6] https://gitlab.apertis.org/appfw/frampton/tree/v0.6.1/scripts
[7] applications.md#permissions
[8] https://martyn.pages.apertis.org/apertis-website/glossary/#system-service

Recommendation

The Apertis Application Bundle Specification[9] describes the fields that can appear in application entry points and are expected to remain supported long-term. This document provides rationale for those fields, suggested future directions, and details of functionality that is not necessarily long-term stable.

App identification

Each built-in or store application bundle has a *bundle ID*, which is a reversed domain name[10] such as `org.apertis.Frampton`.

Each entry point within an application bundle has an *entry point ID*, which is a reversed domain name such as `org.apertis.Frampton.Agent`.

For simple bundles with a single entry point, the bundle ID and the entry point ID should be equal.

For more complex bundles with multiple entry points, the entry point ID should *start with* the bundle ID, but may have additional components.

All names should be allocated in a namespace controlled by the author of the bundle — in particular, Apertis applications should be in `org.apertis`. Sample code that is not intended to be used in production should be placed in `com.example`, with `org.example` and `net.example` also available for code samples that need to demonstrate the interaction between multiple namespaces (we prefer `com.example`, as a hint to developers that reversed domain names do not always start with "org").

Desktop entries

Each Apertis *application entry point* is represented by a standard freedesktop.org Desktop Entry[11] (a `.desktop` file in `XDG_DATA_DIRS/applications`). The desktop file must be named using the entry point ID, so `org.apertis.Frampton.Agent` would have `org.apertis.Frampton.Agent.desktop`.

The localestring[12] mechanism is used for translated strings.

Built-in application bundles install their desktop files in `${prefix}/share/applications`, which expands to `/usr/Applications/${bundle_id}/share/applications`. They also install symbolic links in `/usr/share/applications` pointing to the real files. It is technically possible for any process to read this location.

Store applications install their desktop files in `${prefix}/share/applications`, which expands to `/Applications/${bundle_id}/share/applications`. The app installer is responsible for creating symbolic links in `/var/lib/apertis_extensions/applications`

[9]https://martyn.pages.apertis.org/apertis-website/architecture/bundle-spec/
[10]https://en.wikipedia.org/wiki/Reverse_domain_name_notation
[11]http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html
[12]https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html#localized-keys

pointing to the real files. Only processes with appropriate permissions are allowed to read these locations.

Apertis applications must have the `X-Apertis-Type` key in their metadata, so that they will be listed in Apertis. They should usually also have `OnlyShowIn=Apertis;` so that they do not appear in the XFCE desktop environment menu in SDK images.

The value of the `Exec` key must start with an absolute path to the executable below `${prefix}`. This ensures that the application framework can detect which app-bundle the executable belongs to.

Entry points that would not normally appear in a menu, including all background services (agents), should have `NoDisplay=true`.

The `Interfaces` key is used for Interface discovery[13]. In particular, the following interfaces are defined:

- `org.apertis.GlobalSearchProvider`: Indicates that the application is a global search provider, equivalent to the `supports-global-search` schema entry.

The standard `MimeType` key controls the possible content-type and URI-scheme associations[14]. For example, `x-scheme-handler/http` is used in desktop environments such as GNOME to designate an application as capable of acting as a general-purpose web browser, and we will do the same here. The Didcot service mediates applications' access to this information; for example, it may set priorities or ignore certain applications or associations altogether.

Services that parse desktop files should use the implementation in GLib[15], or an Apertis-specific API built on top of that implementation.

Additional recommended keys

The following additional keys are defined in the `[Desktop Entry]` group.

- `X-Apertis-ParentEntry` (string): For situations where multiple menu entries start the same program in different modes, all but one of those menu entries set `X-Apertis-ParentEntry` to the entry point ID of the remaining menu entry. See Multiple-view applications and the [D-Bus Activation][Bundle spec D-Bus activation] section of the Apertis Application Bundle Specification.

- `X-Apertis-ServiceExec` (string): A command-line similar to `Exec` that starts the entry point in the background, without implicitly *activating* it (causing it to show a window) if it is a graphical program. For example, entry points that use `GApplication` will usually use the same executable as for

[13]https://martyn.pages.apertis.org/apertis-website/concepts/interface_discovery/
[14]https://martyn.pages.apertis.org/apertis-website/concepts/content_hand-over/
[15]https://gitlab.gnome.org/GNOME/glib/-/blob/main/gio/gdesktopappinfo.c#L2007

`Exec` here, but add the `--gapplication-service` option to it. See the [D-Bus Activation][Bundle spec D-Bus activation] section of the Apertis Application Bundle Specification.

- `X-GNOME-FullName` (localestring): The human-readable full name of the application, such as `Rhayader Web Browser`. This key is already used by the GLib library, and by desktop environments based on it (such as GNOME). Like `Name`, this is a "localestring": non-English versions can be provided with syntax like `X-GNOME-FullName[fr]=Navigateur Web Rhyader`.

Potential future keys

The following additional keys have been proposed for the `[Desktop Entry]` group.

- `X-Apertis-BandwidthPriority` (string): Bandwidth priority, currently chosen from `highest`, `high`, `normal`, `low` or `lowest`. As a future extension, numeric priorities could be added, with those strings mapped to reasonable values.

Audio roles

> Requirements-gathering for the audio manager is ongoing. An `X-Apertis-AudioRole` key was initially proposed, but it seems likely that support for specifying a default audio role for PulseAudio streams that do not specify one will be moved from entry points into application bundle metadata[16].

The audio role should have one of the well-known media roles defined by PulseAudio[17].

Additionally, Apertis defines the following roles. Their semantics are not clear, and they should be clarified or deprecated.

- `none`
- `interrupt`
- `record` (possibly the same thing as PulseAudio's `production`, denoting an application that creates or edits audio files, such as a sound recorder)
- `external`
- `unknown`

Additional provisional/deprecated keys

The following provisional keys are defined in the `[Desktop Entry]` group, but are anticipated to be superseded, adjusted or redefined in future.

- `X-Apertis-Type` (string): The application type, chosen from `application`, `service`, `ext-app`, `agent-service`, `startup-application`. Applications with

[16] application-bundle-metadata.md
[17] http://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/Developer/Clients/ApplicationProperties/

no `X-Apertis-Type` are not currently run or displayed in Apertis. This should eventually be replaced with a set of boolean flags describing specific behaviours, such as “start immediately” and “is expected to display a window”; these could either be flags in the file, or indicated in another appropriate way, for example a symbolic link in `/etc/xdg/autostart` for applications and services that should be started immediately.
- `X-Apertis-CategoryLabel` (string; this would normally be a localestring, but the current mildenhall-launcher relies on specific string values for category labels, so translating it is not useful): The name of the menu category. This will be implemented in the short term to keep the current version of Mildenhall-Launcher operational, but should be considered to be deprecated. Instead, launchers should parse the standard `Categories` key, which contains a list of standardized machine-readable categories with the possibility to add Apertis-specific extensions, and translate those into the categories required by the desired UX.
- `X-Apertis-CategoryIcon` (string): The short name of an icon for the category, such as `icon_settings_AC`. In the short term, Canterbury translates this to `/icon_settings_AC.png` to keep the current version of Mildenhall-Launcher operational. Like `X-Apertis-CategoryLabel`, this should be considered to be deprecated; instead, the launcher should determine an icon name from the standard `Categories` key.
- `X-Apertis-BackgroundState` (string): What will happen to the application when it is placed in the background: `running` (i.e. don’t kill), `stopped` (i.e. pause the process), `killed` (i.e. kill the process). This key and its values should ideally be replaced with something that more obviously describes an action rather than a state, such as `kill`, `pause`, `continue`.
- `X-Apertis-DataExchangeRules` (string): This appears to be something to do with Didcot, but its semantics are unclear. The only known example is `default-data`. It should be clarified or dropped.
- `X-Apertis-ManifestUrl` (string): This appears to be intended to point to the JSON manifest for the app bundle, but in the majority of the apps that are currently implemented, it points to a nonexistent XML file, or to the GSettings schema in which it is defined. It should be clarified or dropped.
- `X-Apertis-SplashScreen` (string): None of the current app bundles have this, and it is unclear what its value is meant to be. It is currently passed to the compositor via a D-Bus method call.

Transitional considerations

In addition to `/var/lib/apertis_extensions/applications`, Canterbury reads store app bundles’ entry points from `/var/lib/MILDENHALL_extensions/applications` and `/var/lib/SAC_extensions/applications`, which are two older names for the same thing. We should remove that feature when everything has migrated to `/var/lib/apertis_extensions/applications`.

Canterbury currently has special handling for the executable's arguments:

- An argument named exactly `url` is assumed to be followed by a placeholder; that placeholder is replaced by the actual URL if the application is to be launched with a URL argument. In the short term, this will be preserved. In the longer term, Canterbury and applications should migrate to [the standard `%u`, `%f`, `%U`, `%F` placeholders]Desktop Entry placeholders[18] for a URL, filename, list of URLs or list of filenames respectively.
- An argument named exactly `app-name` is assumed to be followed by a placeholder; that placeholder is replaced by the *entry point ID*. In the short term, this will be preserved. In the longer term, this should be dropped; applications should know their own entry point IDs.
- An argument named exactly `play-mode` is assumed to be followed by a placeholder; that placeholder is replaced by `play` or `stop`. In the short term, this wil be preserved. In the longer term, media player applications should implement Desktop Entry actions[19] instead.

There is currently special handling for several arguments with value exactly `### UNKNOWN ###`. In the long term this should be removed.

In the long term, the category should be replaced by the standard `Categories` key, preferably with values chosen from the XDG Desktop Menu specification[20]. This would allow for variants that do not use precisely the same taxonomy of applications as mildenhall-launcher; because `Categories` is a list, the launcher may use fine-grained categories if desired, falling back to more general top-level categories such as `AudioVideo` if it does not understand any more specific category.

The application launcher HMI should translate these categories into whatever was specified by the variant's UX designer; for example, mildenhall-launcher would translate `Video` to "Video & TV", `Office` to "Productivity", and `Maps` to "Travel". The application launcher HMI should also be responsible for presentational logic such as displaying "Travel" as "T R A V E L" if desired.

Features with no direct replacement

`env-key-value-pair` in the GSettings schemata does not currently appear to be used. We recommend removing this feature: application bundles should normally be written to not need a special environment. If they do need special environment variables, the desktop file could specify a shell script as its `Exec` program, with that shell script setting appropriate environment variables and then `exec`ing the real binary.

[18] http://standards.freedesktop.org/desktop-entry-spec/latest/ar01s06.html#exec-variables

[19] http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html#extra-actions

[20] http://standards.freedesktop.org/menu-spec/latest/apa.html

`tile-thumbnails` in the GSettings schemata does not currently appear to be used. A replacement can be added when the requirements are more clear.

Simple applications (one entry point)

This is the simple case where an entry point has one "view", for example the Rhayader web browser.

We install symlinks in `/usr/share/applications` (for built-in app bundles) or `/var/lib/apertis_extensions/applications` (for store app bundles) pointing to the real file in `{/usr,}/Applications/${bundle_id}/share/applications`, with content similar to this.

```
# /usr/share/applications/org.apertis.Rhayader.desktop
[Desktop Entry]
Type=Application
Name=Rhayader
GenericName=Browser
X-GNOME-FullName=Rhayader Browser
Exec=/usr/Applications/org.apertis.Rhayader/bin/rhayader %U
Path=/usr/Applications/org.apertis.Rhayader
X-Apertis-Type=application
X-Apertis-InternetPriority=normal
Categories=Network;WebBrowser;
MimeType=text/html;x-scheme-handler/http;x-scheme-handler/https;
Icon=applications-internet
```

Services

Services are the same as applications (in particular, they have `Type=Application`), except for these special cases:

- they have `NoDisplay=true` to hide them from the menus
- the `X-Apertis-Type` is `service` or `agent-service`

Entry points which do not appear in the menus

Some bundles might have an entry point that exists only to be started as a side-effect of other operations, for instance to handle URIs and content-types[21]. Those entry points would have `NoDisplay=true` to hide them from the menus; that is the only difference.

Multiple-view applications

Some bundles have more than one entry in the system menus; the example we know about is Frampton. We propose to represent these with one `.desktop` file per menu entry.

[21] https://martyn.pages.apertis.org/apertis-website/concepts/content_hand-over/

In this model, each menu entry is a `.desktop` file. Frampton would install `org.apertis.Frampton.Artists.desktop`, `org.apertis.Frampton.Songs.desktop` and `org.apertis.Frampton.Albums.desktop`. In addition, it would install `org.apertis.Frampton.desktop` with `NoDisplay=true`.

The running instance of Frampton would always identify itself as `org.apertis.Frampton`, and the other three `.desktop` files use `X-Apertis-ParentEntry=org.apertis.Frampton` to link them to that name.

When using [D-Bus activation]Desktop Entry D-Bus Activation[22] for applications (which is recommended), Frampton would have separate D-Bus `.service` files for all four names, would take all four bus names and their corresponding object paths at runtime, and would export the `org.freedesktop.Application` API at all four paths; but all of them would have `SystemdService=org.apertis.Frampton.service` to ensure that only one activation occurs. The `Activate`, `Open` or `ActivateAction` method on each bus name would open the relevant view.

The result would look something like this:

```
# org.apertis.Frampton.desktop
[Desktop Entry]
Type=Application
Name=Frampton
GenericName=Audio Player
X-GNOME-FullName=Frampton Audio Player
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton %F
Path=/usr/Applications/org.apertis.Frampton
X-Apertis-Type=application
Categories=Audio;Player;Music;
MimeType=audio/mpeg;
NoDisplay=true;
Icon=music
X-Apertis-ServiceExec=/usr/Applications/org.apertis.Frampton/bin/frampton --
gapplication-service

# org.apertis.Frampton.Artists.desktop
[Desktop Entry]
Type=Application
Name=Frampton — Artists
GenericName=Artists
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --artists
Path=/usr/Applications/org.apertis.Frampton
X-Apertis-Type=application
Categories=Audio;Player;Music;
Icon=music-artist
```

[22] http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html#dbus

```
X-Apertis-ParentEntry=org.apertis.Frampton

# org.apertis.Frampton.Albums.desktop
[Desktop Entry]
Type=Application
Name=Frampton — Albums
GenericName=Albums
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --albums
Path=/usr/Applications/org.apertis.Frampton
X-Apertis-Type=application
Categories=Audio;Player;Music;
Icon=music-album
X-Apertis-ParentEntry=org.apertis.Frampton

# org.apertis.Frampton.Songs.desktop
[Desktop Entry]
Type=Application
Name=Frampton — Songs
GenericName=Songs
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --songs
Path=/usr/Applications/org.apertis.Frampton
X-Apertis-Type=application
Categories=Audio;Player;Music;
Icon=music-track
X-Apertis-ParentEntry=org.apertis.Frampton
```

Appendix: GSettingsSchema-based entry point registration prior to October 2015

As of early October 2015, Canterbury uses GSettings schemata for entry point registration. This is not an intended use of GSettings — the existence of an entry point is not a setting — and it should be avoided.

Canterbury reads the schemata from the default system paths, and from a configurable path (`p_app_manager_get_store_apps_schema_path()`) which in practice resolves to `/Applications/System/registry`. For each schema in the path, if the name does not start with either `com.app` (this prefix is actually configurable, `project-domain`) or `org.secure_automotive_cloud.service`, then the schema is ignored.

Proposed replacement: Canterbury reads desktop files from at least `/var/lib/apertis_extensions/applications` and `/usr/share/applications`, and may read additional locations if desired. This should be done by setting Canterbury's `XDG_DATA_DIRS` to include at least `/var/lib/apertis_extensions` and `/usr/share`.

Canterbury reads the following keys, each with a corresponding constant such as `APP_NAME` except where noted:

- `app-name` (`pAppName`): A string: entry point ID, such as `frampton`, `Frampton-Agent`. *Proposed replacement: the name of the `.desktop` file.*
- `background-state` (`uinBkgState`): One of { `running`, `stopped`, `killed`, `unknown` }. *Proposed replacement: `X-Apertis-BackgroundState`*
- `working-directory` (`pWorkingDirectory`): A string: the app's initial working directory, which in practice must be in its directory `/usr/Applications/xyz` for built-in apps (Ribchester assumes this, and uses it to create the app's storage during first-boot). *Proposed replacement: the standard `Path` key.*
- `exec-path` (`pExecutablePath`): A string: the executable. *Proposed replacement: the first word of the standard `Exec` key.*
- `exec-type` (`uinExecutableType`): One of { `application`, `service`, `ext-application` (or sometimes `ext-app`, depending on project), `agent-service`, `unknown` }. *Proposed replacement: `X-Apertis-Type`.*
- `exec-args` (`pExecutableArgv`): An array of (string, string) pairs which are flattened into a single list for `exec()`, for example `[('app-name', 'AudioPlayer'), ('menu-entry', 'A R T I S T S'), ('url', ' ')]` turns into executing the equivalent of the Python code `subprocess.call(['/usr/Applications/frampton/bin/frampton', 'app-name', 'AudioPlayer', 'menu-entry', 'A R T I S T S', 'url', ' '])`. *Proposed replacement: the standard `Exec` key, except for its first word.*
- `internet-bw-prio` (`uinInternetBandwidthPriority`): One of { `highest`, `high`, `mid`, `low`, `lowest`, `unknown` } or unspecified. *Proposed replacement: `X-Apertis-BandwidthPriority`. Additionally, we recommend accepting `normal` as a synonym for `mid`.*
- `splash-screen` (`pSplashScreen`): A string. No application specifies this, so we do not know what its purpose is. *Proposed replacement: `X-Apertis-SplashScreen`, or remove the feature.*
- `audio-resource-type` (`CANTERBURY_AUDIO_RESOURCE_TYPE`, `uinAudioResourceType`): A `CanterburyAudioType`. *Proposed replacement: use PulseAudio stream roles.*
- `audio-channel-name` (`pAudioChannelName`): A string: the name of the audio channel. *Proposed replacement: make the audio manager derive the bundle ID from the AppArmor profile in a way that cannot be faked by a malicious app-bundle.*
- `audio-resource-owner` (`CANTERBURY_AUDIO_RESOURCE_OWNER`, `pAudioResourceOwner`): A string: the entry point ID of the entry point that will generate audio on behalf of this HMI. *Proposed replacement: make the audio manager derive the bundle ID from the AppArmor profile in a way that cannot be faked by a malicious app-bundle.*
- `category` (`pCategory`): A string: the displayed name of the category. *Proposed replacement: `X-Apertis-CategoryLabel` in the short term, `Categories` in the longer term.*
- `category-icon` (`pCategoryIcon`): A string: the name of the category icon, of the form `/icon.png`, which appears to be relative to the launcher's data directory. *Proposed replacement: `Icon`, changing the value to be defined to be found via the freedesktop.org icon theme specification.*

- `env-key-value-pair` (`ENV_KEY_VALUE`, `pEnvKeyValuePair`): An array of strings, of even length: the environment of the subprocess. *Proposed replacement: remove.*
- `window-name` (`APP_WIN_NAME`, `pAppWinName`): A string: the name of the window that this HMI is expected to map. *Proposed replacement: make the compositor derive the bundle ID from the AppArmor profile in a way that cannot be faked by a malicious app-bundle.*
- `application-entry-names` (`APP_ENTRY_NAME_LIST`, `pApplicationEntryName`): An array of strings. Each one is the title of a quick-menu (right panel) entry in `mildenhall-launcher`. The main-menu (left panel) entry is taken from `category` and `category-icon`. *Implemented replacement: one desktop file per entry point, and use its `X-GNOME-FullName`, `GenericName` and/or `Name`. The ability to have more than one menu entry per application is replaced by `X-Apertis-ParentEntry`.*
- `application-entry-icons` (`APP_ENTRY_ICON_LIST`, `pApplicationEntryIcon`): An array of strings: `file:///` URLs to icons, in the same order as `application-entry-names`. *Implemented replacement: `Icon` may name either an icon in the icon theme, or a `file:///` URL.*
- `tile-thumbnails` (`APP_ENTRY_TILE_LIST`, `pApplicationTileThumbnail`): An array of strings that represent home-screen tiles in some unspecified way; we do not have any examples to use for reference. *Proposed replacement: behave as though no application has a home-screen tile, and design home-screen tiles separately.*
- `manifest-url` (`MANIFEST_FILE_URL`, `pAppManifestUrl`): A string representing the manifest in some way. *Proposed replacement: `X-Apertis-ManifestUrl`, or remove; services should find desktop files for entry points in the standard way, and should find manifests for app-bundles by looking in well-known locations for files whose names are based on the bundle ID.*
- `app-settings-icon` (`pAppSettingsIcon`): A string representing an icon used for settings. *Proposed replacement: icon from application bundle metadata*[23].
- `app-settings-name` (`pAppSettingsName`): A string representing a label used for settings? *Proposed replacement: name from application bundle metadata*[24].
- `app-settings-path` (`pAppSettingsPath`): A string representing a GSettings hierarchy used for settings. *Implemented replacement: the settings schema (if any) whose name matches the bundle ID appears in the system preferences UI; other schemas do not appear.*
- `mime-type` (`pMimeType`): An array of strings representing content-types that this application can handle, and/or pseudo-content-types such as `mt_app_settings`. *Implemented replacement: `MimeType` for content-type and URL-scheme handlers; `Interfaces` to discover other functionality.*
- `mime-list` (`pMimeList`): An array of strings representing some facet of con-

[23] application-bundle-metadata.md
[24] application-bundle-metadata.md

tent type handling, with values such as `url`, `audio/mpeg` and `launch`. *Proposed replacement: discover feature support with `Interfaces`.*

- `data-exchange-rules` (`DATA_EXCHANGE_FILE`, `pDataExchangeFile`): A string which has something to do with data exchange, with the only known value being `default-data`. *Proposed replacement: none.*
- `supports-global-search` (`SUPPORT_GLOBAL_SEARCH`, `bSupportsGlobalSearch`): A boolean value indicating support for acting as a global search provider. *Proposed replacement: if this would have been true, then `org.apertis.GlobalSearchProvider` appears in `Interfaces`.*

Appendix: other approaches to multiple-view applications

We considered some other approaches to this feature.

One `Desktop Action` per view

In this model, each entry point (application or service) is a `.desktop` file. Frampton would install `org.apertis.Frampton.desktop`, with contents something like this:

```
# org.apertis.Frampton.desktop
[Desktop Entry]
Type=Application
Name=Frampton
GenericName=Audio Player
X-GNOME-FullName=Frampton Audio Player
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton %F
Path=/usr/Applications/org.apertis.Frampton
X-Apertis-Type=application
X-Apertis-AudioRole=music
X-Apertis-AudioChannelName=org.apertis.Frampton.Agent
Categories=Audio;Player;Music;
MimeType=audio/mpeg;
NoDisplay=true;
Actions=albums;artists;songs;
Icon=music

[Desktop Action artists]
Name=Artists
Icon=music-artist
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --artists
X-Apertis-ShowInMenu=true

[Desktop Action albums]
Name=Albums
Icon=music-album
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --albums
```

```
X-Apertis-ShowInMenu=true

[Desktop Action songs]
Name=Songs
Icon=music-track
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --songs
X-Apertis-ShowInMenu=true

# this is *not* a "quick menu" entry
[Desktop Action shuffle]
Name=Shuffle All
Icon=music-shuffle
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton-control   --shuffle-
all
```

The Desktop Entry Specification specifies that application launchers should present desktop actions[25] to the user within the context of an application, for instance as a submenu, but that isn't how the UX of `mildenhall-launcher` works. We therefore use `X-Apertis-ShowInMenu` to indicate that these particular desktop actions should be made available to the user even though their parent `org.apertis.Frampton` is not.

This could be combined with desktop actions as specified in the Desktop Entry Specification if desired; those desktop actions would simply omit `X-Apertis-ShowInMenu`. For example, if it was desirable for a long press on Frampton's menu entries to result in a menu of actions such as "shuffle all", "import from USB drive", "buy music", then those could be represented as desktop actions.

One Apertis-specific menu entry per view

This model is similar to the one with desktop actions, but it acknowledges that desktop actions were not really designed to work that way, and uses Apertis-specific syntax inspired by desktop actions instead:

```
# org.apertis.Frampton.desktop
[Desktop Entry]
Type=Application
Name=Frampton
GenericName=Audio Player
X-GNOME-FullName=Frampton Audio Player
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton %F
Path=/usr/Applications/org.apertis.Frampton
X-Apertis-Type=application
X-Apertis-AudioRole=music
X-Apertis-AudioChannelName=org.apertis.Frampton.Agent
```

[25]http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html#extra-actions

```
Categories=Audio;Player;Music;
MimeType=audio/mpeg;
X-Apertis-MenuEntries=albums;artists;songs;
Icon=music

[Apertis Menu Entry artists]
Name=Frampton — Artists
GenericName=Artists
Icon=music-artist
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --artists

[Apertis Menu Entry albums]
Name=Frampton — Albums
GenericName=Albums
Icon=music-album
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --albums

[Apertis Menu Entry songs]
Name=Frampton — Songs
GenericName=Songs
Icon=music-track
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton --songs

[Desktop Action shuffle]
Name=Shuffle All
Icon=music-shuffle
Exec=/usr/Applications/org.apertis.Frampton/bin/frampton-control   --shuffle-
all
```