Application bundle metadata

Contents

This document extends the Apertis Applications concept design[1] to cover metadata about application bundles[2] (app-bundles).

Terminology and concepts

See the Apertis glossary[3] for background information on terminology. Apertis-specific jargon terms used in this document are hyperlinked to that glossary.

Use cases

These use-cases are not exhaustive: we anticipate that other uses will be found for per-application-bundle metadata. At the time of writing, this document concentrates on use-cases associated with assigning priorities to requests from an app-bundle to a platform service.

[1] https://sjoerd.pages.apertis.org/apertis-website/concepts/applications/

[2] https://sjoerd.pages.apertis.org/apertis-website/glossary/#application-bundle

[3] https://sjoerd.pages.apertis.org/apertis-website/glossary/

Audio management priorities

Assume that the Apertis audio management component (which is outside the scope of this document) assigns priorities to audio streams based on OEM[4]-specific rules, potentially including user configuration.

Suppose the author of an app-bundle has a legitimate reason to have their audio streams played with an elevated priority, for example because their app-bundle receives voice calls which should take precedence over music playback.

Also suppose a different, malicious app-bundle author wishes to interrupt the driver's phone call to play an advertisement or other distracting sound as an audio stream.

The Apertis system must be able to distinguish between the two app-bundles, so that requests for an elevated priority from the first app-bundle can be obeyed, while requests for an elevated priority from the second app-bundle are rejected.

We assume that the app-bundles have been checked by an app-store curator before publication, and that the first app-bundle declares a special permission[5] in its app manifest, resulting in the app framework allowing it to flag its audio stream in ways that will result in it being treated as important, and hence superseding less important audio. Conversely, if the second app-bundle had declared that permission, we assume that the app-store curator would have recognised this as inappropriate and reject its publication.

Notification and dialog priorities

Assume that the Apertis compositor (which is outside the scope of this document) assigns priorities to notifications based on OEM[6]-specific rules, potentially including user configuration. Depending on the OEM's chosen UX design, app-modal and system-modal dialogs might be treated as visually similar to notifications; if they are, the compositor author might also wish to assign priorities from the same ranges to dialogs.

Similar to the Audio management priorities use case, app-bundles that have a legitimate reason for their notifications or dialogs to be high-priority must be able to achieve this, but malicious app-bundles whose authors aim to misuse this facility must not be able to achieve an elevated priority.

App-bundle labelling

A UX designer might wish to arrange for all user interface elements associated with a particular app-bundle (including notifications, windows, its representation in lists of installed app-bundles, and so on) to be marked with an unam-

[4] https://sjoerd.pages.apertis.org/apertis-website/glossary/#oem
[5] https://sjoerd.pages.apertis.org/apertis-website/designs/permissions/
[6] https://sjoerd.pages.apertis.org/apertis-website/glossary/#oem

biguous indication of the app-bundle that created them, such as its name and icon.

In particular, the Compositor Security concept design (which is work in progress[7] at the time of writing) calls for windows and notifications to be visually associated with the app-bundle that created them, so that malicious app-bundle authors cannot make the user believe that information presented by the malicious app-bundle came from a different app-bundle (*output integrity*), and also cannot convince the user to enter input into the malicious app-bundle that they had only intended to go to a different app-bundle (a *trusted input path*, providing *input confidentiality* for the non-malicious app-bundle).

Note this mechanism will not be effective unless either the app-store curator avoids accepting app-bundles with the same or confusingly similar names or icons, or the UX designer disambiguates app-bundles using something that is guaranteed to be unique, such as the app-bundle ID (which is not necessarily a desirable or user-friendly UX). This applies wherever app-bundles are listed, such as the app store's on-device user interface, the app-store's website, or a list of installed app-bundles in the device's equivalent of Android's Settings → Apps view.

Requirements

App-bundle metadata

An Apertis platform library to read app bundle metadata must be made available to platform components, featuring at least these API calls:

- given a bundle ID, return an object representing the metadata
- list all installed bundles (either built-in or store) with their IDs and metadata
- emit a signal whenever the list of installed bundles changes, for example because a store app bundle was installed, removed, upgraded or rolled back (simple change-notification)

Labelling requirements

Each app-bundle must contain a human-readable name in international English. It must also be possible for an app-bundle to contain translated versions of this name for other languages and locales, with the international English version used in locales where a translation is not provided.

Each app-bundle must be able to contain the name of the authoring company or individual.

Each app-bundle must contain a version number. To let the application manager make appropriate decisions, all application bundles must a format for their

[7] https://sjoerd.pages.apertis.org/apertis-website/concepts/compositor_security/

version strings that can be compared in a standard way. How an application developer chooses to set the version numbers is ultimately their decision, but Apertis must be able to determine whether one version number is higher than another.

Collabora recommends requiring version numbers to be dotted-decimal (one or more decimal integers separated by single dots), with "major.minor.micro" (for example `3.2.4`) recommended but not strictly required.

There will be a "store version" appended to the version string after a dash, similar to the versioning scheme used by `dpkg`; for example, in the version string `3.2.4-1`, the `1` is the store version. The store version allows the store to push an update even if the application version hasn't changed, and it will be the lowest significant figure. For example, version `2.2.0-1` is newer than version `2.1.99-4`. The store version will re-start at 1 any time the application version is increased, and will be incremented if a new intermediate release is required.

Secure identification

Apertis platform[8] services that receive requests from an unknown process must be able to identify which app-bundle the process belongs to. To support this, the request must take place via a channel that guarantees integrity for that process's identification: it must not be possible for a malicious process to impersonate a process originating from a different app-bundle.

Audio stream and notification requirements

The information required by the audio manager must be represented as one or more metadata key-value pairs that can be read from the app bundle metadata.

The information required by the notification implementation must be represented as one or more metadata key-value pairs that can be read from the app bundle metadata.

We anticipate that audio management and notifications will not always assign the same priority to each app-bundle, therefore it must be possible for the metadata keys used by audio management and those used by notifications to be distinct.

App-store curator oversight

It must be straightforward for an app-store curator to inspect the metadata that is present in an app-bundle, for example so that they can refuse to publish app-bundles that ask for audio or notification priorities that they have no legitimate reason to use, or for which the name, icon or other information used for App-bundle labelling is misleading.

[8] https://sjoerd.pages.apertis.org/apertis-website/glossary/#platform

Store app-bundle confidentiality

Ordinary unprivileged programs in store app-bundles must not be able to use these API calls to enumerate other installed store app-bundles. For example, if those API calls are implemented in terms of a D-Bus service, it must reject method calls from store app-bundles, or if those API calls are implemented in terms of reading the filesystem directly, store app-bundles' AppArmor profiles must not allow reading the necessary paths.

Non-requirement: it is acceptable for ordinary unprivileged programs to be able to enumerate installed built-in app-bundles. Built-in app-bundles are part of the platform, so there is no expectation of confidentiality for them.

Extension points

We anticipate that vendors will wish to introduce non-standardized metadata, either as a prototype for future standardization or to support vendor-specific additional requirements. It must be possible to include new metadata fields in an app-bundle, without coordination with a central authority. For example, this could be achieved by namespacing new metadata fields using a DNS name (as is done in D-Bus[9]), namespacing them with a URI (as is done in XML[10]), or using the `X-Vendor-NewMetadataField` convention[11] (as is done in email headers, HTTP headers and freedesktop.org `.desktop` files[12]).

Future directions

Platform API requirements

The application bundle metadata should include a minimum system version (API version) required to run the application, for example to prevent the installation of an application that requires at least Apertis 16.12 in an Apertis 16.09 environment. A specific versioning model for the Apertis API has not yet been defined.

Declarative permissions

The application bundle metadata should include simple, declarative permissions[13] which can be used to generate an AppArmor profile in an automated way. The Permissions concept design[14] tracks this work.

[9] https://dbus.freedesktop.org/doc/dbus-specification.html#message-protocol-names

[10] https://www.w3.org/TR/REC-xml-names/

[11] https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html#extending

[12] https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html

[13] https://sjoerd.pages.apertis.org/apertis-website/designs/permissions/

[14] https://sjoerd.pages.apertis.org/apertis-website/designs/permissions/

Declaring system extensions

The Applications concept design[15] calls for app-bundle metadata to describe the types of system extension[16] (themes, addons, plugins, etc.) provided by an app-bundle. There is currently no detailed specification for this.

AppStream upstream XML already supports declaring that a component (app-bundle) is an addon to another component (via the `addon` type) or to the system as a whole (via the `<provides>` element). There is no specific metadata to describe themes; discussion has been started in AppStream issue 67[17].

Declaring where applications store non-essential files

The Applications concept design[18] suggests that application bundle metadata might declare where applications store non-essential files, so that the system can delete those files when disk space runs out.

Placing symbolic links in centralized locations

The Applications concept design[19] suggests that for applications not originally designed for Apertis, which might write to locations like `~/.someapp`, application bundle metadata might declare where the platform must create symbolic links to cause those applications to read and write more appropriate locations on Apertis, for example `~/.someapp` → `/var/Applications/com.example.SomeApp/users/${UID}/data`.

Declaring an EULA

App-bundle metadata should include a way to specify an EULA which the user must agree with before the application bundle will be installed. See AppStream issue 50[20] for work on this topic in the AppStream specification.

Other files in the license directory of the bundle but not mentioned in this way will still be copied the device, and the HMI components must provide some way to view that information later.

Placeholder icons

Since the installation process is not instant, a placeholder icon should be provided and specified in the version of the application bundle metadata that is downloaded from the application store. This icon will be copied into the store directory by the application store during publication. It will be displayed by

[15] https://sjoerd.pages.apertis.org/apertis-website/concepts/applications/
[16] https://sjoerd.pages.apertis.org/apertis-website/concepts/applications/#system-extensions
[17] https://github.com/ximion/appstream/issues/67
[18] https://sjoerd.pages.apertis.org/apertis-website/concepts/applications/
[19] https://sjoerd.pages.apertis.org/apertis-website/concepts/applications/
[20] https://github.com/ximion/appstream/issues/50

the application manager instead of the application until the installation is completed. The application launcher will also be able to display a progress indicator or – if multiple applications are being installed – a position in the install queue.

Platform component metadata

Although it is not a requirement at this stage, we anticipate that it might be useful in the future to be able to associate similar metadata with platform components, such as the Newport download manager.

Other systems

This section contains a very brief overview of the analogous functionality in other open-source platforms.

freedesktop.org AppStream

Several open-source desktop platforms such as GNOME and KDE, and Linux distributions such as Ubuntu and Fedora, have adopted AppStream[21] as a shared format for software component metadata, complementing the use of `.desktop` files[22] for entry points[23].

The AppStream specification refers to *components*, which are a generalization of the same concept as Apertis app-bundles, and can include software from various sources, including traditional distribution packages and bundling technologies such as Flatpak.

Flatpak

The Flatpak[24] framework provides user-installable applications analogous to Apertis app-bundles. It uses AppStream[25] for app-bundle metadata, together with `.desktop` files[26] for entry points.

Snappy

Ubuntu Snappy[27] packages (snaps) are also analogous to Apertis app-bundles. Their metadata[28] consists of a Snappy-specific YAML file describing the snap,

[21] https://www.freedesktop.org/software/appstream/docs/

[22] https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html

[23] https://sjoerd.pages.apertis.org/apertis-website/concepts/application-entry-points/

[24] http://flatpak.org/

[25] https://www.freedesktop.org/software/appstream/docs/

[26] https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html

[27] http://snapcraft.io/

[28] https://developer.ubuntu.com/en/snappy/guides/meta/

again together with `.desktop` files[29] describing entry points.

Android

Android *apps* are its equivalent of Apertis app-bundles. Each app has a single App manifest[30] file, which is an XML file with Android-specific contents, and describes both the app itself, and any *activities* that it provides (activities are analogous to Apertis entry points[31]).

Design recommendations

The Apertis Application Bundle Specification[32] describes the metadata fields that can appear in an application bundle 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-bundle metadata design

We anticipate that other designs involving app-bundles will frequently require other metadata beyond the use-cases currently present in this document, for example categories. As such, we recommend introducing a general metadata file into built-in and store app-bundles.

This metadata file could have any syntax and format that is readily parsed. To minimize duplicate effort, we recommend using AppStream XML[33], a format designed to be shared between desktop environments such as GNOME and KDE, and between Linux distributions such as Ubuntu and Fedora.

Each built-in app bundle should install an AppStream upstream XML[34] metadata file. If the built-in app bundle has entry points[35], then its metadata file must be made available as `/usr/share/metainfo/${bundle_id}.appdata.xml` (where `${bundle_id}` represents its bundle ID), and its `<id>` must be `<id type="desktop">${entry_point_id}.desktop</id>` where `${entry_point_id}` represents its primary entry point (typically the same as the bundle ID). `/usr/share/metainfo/${bundle_id}.appdata.xml` will typically be a symbolic link to `/usr/Applications/${bundle_id}/share/metainfo/${bundle_id}.appdata.xml`.

If the built-in app bundle has no entry points (for example a theme), then its metadata file must be available as `/usr/share/metainfo/${bundle_id}.metainfo.xml` (where `${bundle_id}` represents its bundle ID), and its `<id>` must be the same as

[29] https://specifications.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html

[30] https://developer.android.com/guide/topics/manifest/manifest-intro.html

[31] https://sjoerd.pages.apertis.org/apertis-website/concepts/application-entry-points/

[32] https://sjoerd.pages.apertis.org/apertis-website/architecture/bundle-spec/

[33] https://www.freedesktop.org/software/appstream/docs/

[34] https://www.freedesktop.org/software/appstream/docs/chap-Metadata.html

[35] https://sjoerd.pages.apertis.org/apertis-website/concepts/application-entry-points/

its bundle ID. Again, this would typically be a symbolic link to a corresponding path in `/usr/Applications/${bundle_id}`.

Each store app bundle should install an AppStream upstream XML metadata file into `/Applications/${bundle_id}/share/metainfo/${bundle_id}.appdata.xml` or `/Applications/${bundle_id}/share/metainfo/${bundle_id}.metainfo.xml` (depending on whether it has entry points), with contents corresponding to those specified for built-in app bundles. For Store app-bundle confidentiality, a store app-bundle's AppArmor profile must not allow it to read the contents of a different store app-bundle, and in particular its AppStream metadata.

AppStream upstream XML is normally also searched for in the deprecated path[36] `/usr/share/appdata`, but for simplicity, we do not require the `share/appdata/` directory to be processed for application bundles. Since existing application bundles do not contain it, this does not create a compatibility problem.

For App-store curator oversight, if the implementation reads other sources of metadata from a store app-bundle (for example the `.desktop` entry points provided by the app-bundle), then the implementation must document those sources. The app-store curator must inspect all of those sources. This requirement does not apply to built-in app-bundles, which are assumed to have been checked thoroughly by the platform vendor at the time the built-in app-bundle was integrated into the platform image.

The Apertis platform must provide cache files whose timestamps change whenever there is a change to the set of store or built-in app bundles, or to those bundles' contents. These cache files should be monitored by the libcanterbury-platform[37] library, using the standard `inotify` mechanism. Any cache files that contain store app-bundles must not be readable by a store app-bundle, to preserve Store app-bundle confidentiality.

The other APIs that are required are straightforward to implement in the libcanterbury-platform[38] library by reading from the cache files. Because this is done in a library (as opposed to a D-Bus service), the implementation of these APIs will run with the privileges of the process that is requesting the information: in particular, if an unprivileged process attempts to read the cache files, this will be prevented by its AppArmor profile, regardless of whether it is using libcanterbury-platform or reading the files directly.

We recommend that store app-bundles and built-in app-bundles appear in separate cache files, for several reasons:

- In the current design for Apertis operating system upgrades, the metadata files for built-in app-bundles and platform components in

[36] https://www.freedesktop.org/software/appstream/docs/chap-Metadata.html#spec-component-location

[37] https://gitlab.apertis.org/appfw/canterbury

[38] https://gitlab.apertis.org/appfw/canterbury

/usr/Applications/*/share/* and /usr/share/* are only updated during an operating system upgrade, by either dpkg or by unpacking a new OS filesystem hierarchy that will be activated after the next reboot. In the dpkg case, it is sufficient to have a dpkg trigger monitoring these directories, and update the built-in app-bundle cache when they have changed, leaving the store app-bundle cache unchanged. Similarly, in the whole-OS upgrade case, the built-in app-bundle cache can be provided in the new OS filesystem or rebuilt during startup, again leaving the store app-bundle cache unchanged.

- Conversely, the metadata files for store app-bundles are updated by the Ribchester subvolume manager when it installs a new store app-bundle, which can occur at any time. When it does this, it is sufficient to update the store app-bundle cache, leaving the built-in app-bundle cache unchanged.
- If Apertis moves to a more static model for deployment of the platform (for example using disk images or OSTree to deploy pre-built filesystem hierarchies), the built-in app-bundle cache would be entirely static and could be included in the pre-built filesystem hierarchy.
- Using separate caches makes it straightforward to ensure that if a store app-bundle with the same name as a built-in app-bundle is somehow installed, the built-in app-bundle takes precedence.

Any metadata keys and values that have not been standardized by the AppStream project (for example audio roles that might be used to determine a bundle's audio priority) must be represented using Extension points within the AppStream metadata. The formal AppStream specification[39] does not provide an extension point, but the reference implementation[40] and appstream-glib[41] both provide support for a <custom> element with <value> children. We recommend using that element for extension points. See the Apertis Application Bundle Specification[42] for details.

When a store or built-in app-bundle is added, removed or changed, the Apertis platform must update the corresponding cache file.

Future directions

AppStream XML is equally applicable to platform components, which can install metadata in /usr/share/metainfo in the same way as built-in app-bundles.

Because built-in app-bundles and platform components have the same update schedule and are managed by the same vendor (they are both part of the plat-

[39] https://www.freedesktop.org/software/appstream/docs/

[40] https://www.freedesktop.org/software/appstream/docs/api/index.html

[41] https://github.com/hughsie/appstream-glib/

[42] https://sjoerd.pages.apertis.org/apertis-website/architecture/bundle-spec/

form), we anticipate that platform components should use the same cache file as built-in app-bundles.

Secure identification design

Consumers of requests from app-bundles, such as the audio manager or the notifications implementation, must receive the bundle ID alongside the request using a trusted mechanism. If the request is received via D-Bus, the bundle ID must be retrieved by using the GetConnectionCredentials[43] method call to receive the AppArmor context, then parsing the context to get the bundle ID and whether it is a store or built-in app-bundle. If the request takes the form of a direct `AF_UNIX` socket connection, the bundle ID must be retrieved by reading the `SO_PEERCRED` socket option, then parsed in the same way. Consumers of app-bundle priorities should do this by using the CbyProcessInfo[44] objects provided by libcanterbury[45].

Because the Apertis Security concept design[46] does not place a security boundary between different processes originating from the same app-bundle, all identification of app-bundles should be carried out using their bundle IDs. In particular, consumers of requests from app-bundles should only use the requester's AppArmor label to derive its bundle ID and whether it is a store or built-in app-bundle, and must not use the complete AppArmor label, the complete path of the executable or the name of the corresponding entry point[47] in access-control decisions.

Labelling design

AppStream upstream XML[48] already contains standardized metadata fields for a name, author name etc.

The name (and several other metadata fields) can be translated via the `xml:lang` attribute. For example, GNOME Videos (Totem) has many language-specific names, starting with:

```
<name>Videos</name>
<name xml:lang="af">Video's</name>
<name xml:lang="ar">فيديو</name>
<name xml:lang="as">□□□□□'□□□□</name>
<name xml:lang="be">Відэа</name>
```

[43] https://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-get-connection-credentials
[44] https://gitlab.apertis.org/appfw/canterbury/blob/master/canterbury/process-info.h
[45] https://gitlab.apertis.org/appfw/canterbury
[46] https://sjoerd.pages.apertis.org/apertis-website/designs/security/
[47] https://sjoerd.pages.apertis.org/apertis-website/concepts/application-entry-points/
[48] https://www.freedesktop.org/software/appstream/docs/chap-Metadata.html

AppStream upstream XML does not include an icon, although the derived AppStream collection XML[49] format published by redistributors does. We recommend that the app-bundle should contain a PNG icon whose name matches its bundle ID, installed to its `share/` directory as part of the `hicolor` fallback theme.

> The reserved icon theme name `hicolor` is used as the fallback whenever a specific theme does not have the required icon, as specified in the freedesktop.org Icon Theme specification[50]. The name `hicolor` was chosen for historical reasons.

For example, `com.example.ShoppingList` would include `/Applications/com.example.ShoppingList/share/icons/hicol` If the app-store uses AppStream collection XML, then the process used to build AppStream collection XML from individual applications' AppStream upstream XML files should assume this icon name and include it in the collection XML.

Open question: We should require a specific size for the icon, to avoid blurry or blocky app icons caused by resizing. GNOME Software uses 64×64 as its baseline requirement, but recommends larger icons, for example 256×256. iOS[51] uses 1024×1024 for the App Store and ranges from 60×60 to 180x180 for on-device icons. [Android][Android icons sizes] uses 512×512 for the Google Play Store and ranges from 36×36 to 96×96 for on-device icons. What are our preferred sizes?

Future directions

Platform components that are not part of an app-bundle do not have bundle IDs. We anticipate that Platform component metadata might be identified by a separate identifier in the same reversed-DNS namespace, and that the consumer of requests might derive the platform component identifier by looking for components that declare metadata fields matching the requester's AppArmor label (part of the AppArmor context).

Summary

- app-bundle metadata is read from the cache that summarizes built-in and store app-bundles. The libcanterbury-platform[52] library provides the required APIs; in particular, change notification can be done using `inotify`.
- Secure identification is provided by [parsing the requesting process's AppArmor context][Secure identification design].
- The Audio stream and notification requirements are addressed by providing their desired metadata in the app-bundle metadata, in the form of arbitrary key/value pairs.

[49] https://www.freedesktop.org/software/appstream/docs/chap-CollectionData.html

[50] http://standards.freedesktop.org/icon-theme-spec/icon-theme-spec-latest.html

[51] https://developer.apple.com/library/safari/documentation/UserExperience/Conceptual/MobileHIG/IconMatrix.html

[52] https://gitlab.apertis.org/appfw/canterbury

- App-store curator oversight is facilitated by documenting all of the sources within a store app-bundle from which the implementation gathers metadata to populate its cache.
- Store app-bundle confidentiality is provided by storing the cache file describing installed store app-bundles in a location where store app-bundles cannot read it, and by avoiding the need to introduce a D-Bus service from which they could obtain the same information.
- The appstream-glib[53] library supports Extension points in AppStream XML.

[53] https://github.com/hughsie/appstream-glib/