



Interface discovery

Contents

2	Use cases	2
3	In other systems	3
4	Security considerations	4
5	Restricting who can advertise a given interface	4
6	Communication between consumers and implementors	4
7	Visibility of applications to other applications	4
8	Recommendation	4
9	Selecting a preferred implementation	5
10	Enabling/disabling providers	6
11	Restricting who can advertise a given interface	6
12	Communication between consumers and implementors	7
13	Visibility of applications to other applications	7

14 Various features on Apertis require a way to discover the applications and/or
15 agents that implement a particular set of functionality. We refer to the “API
16 contract” for this set of functionality as an *interface*.

Use cases

- 18 • A [global search user interface](#)¹ requires a list of services that can act as
19 “Auxiliary Sources”(see §6.2 in the Global Search design document). For
20 example, a Spotify client might register itself as a search provider so that
21 searching for a term in a global search will find artists or songs matching
22 that term.
- 23 • An application that will display a [Sharing](#)² menu similar to the one in
24 Android requires a list of applications with which files or data can be
25 shared.
- 26 • A navigation app, potentially from an app-store, obtains [points of inter-](#)
27 [est](#)³ from a number of providers, again potentially from an app-store. In
28 a “pull”model, the navigation app would consume the interface “points-
29 of-interest provider”by sending queries to the implementors and getting
30 results back, and the points-of-interest providers would implement that
31 interface. Conversely, in a “push”model, the navigation app could im-
32 plement the interface “points-of-interest sink”, and the points-of-interest
33 providers could consume that interface by sending points of interest to
34 each sink.
- 35 • If more than one navigation app is installed (for example because an Aper-
36 tis system includes the OEM’s own simple navigation solution, but it is
37 possible to install premium navigation software from the app-store), a set-
38 tings user interface to select the preferred navigation app might need to

¹[/images/apertis-global-search-design-0.3.2.pdf](#)

²https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/sharing/

³https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/points_of_interest/

list all the possible navigation apps.

- Interface discovery could potentially be used with the interface “is the preferred navigation app” to start the navigation app on-demand. If it is, it must be possible to mark one as preferred.
- A navigation app could have a preferences dialog in which [points of interest](#)⁴ providers can be selected or deselected. It should not display points of interest from deselected providers, and should not waste system resources on receiving points of interest from those providers. However, if another application also consumes points of interest, disabling a points of interest provider in the navigation app should not prevent it from being used by the other application.
- The platform could have a preferences dialog in which [points of interest](#)⁵ providers can be selected or deselected. If a POI provider is deselected here, POI consumers such as the navigation app should behave as though the deselected provider had not been installed at all.

In other systems

GNOME Shell’s search provider API⁶ relies on applications registering their support for the search provider “interface” by installing files in `/usr/share/gnome-shell/search-providers`. This is not ideally suited to a platform like Apertis with a strong division between the “platform” and “app bundle” layers, and does not generalize trivially (each interface would have to define its own location in which to place metadata files).

The [freedesktop.org Desktop Entry specification](#)⁷ shared by GNOME, KDE and other open source desktop environments uses `.desktop` metadata files to store metadata about applications. It defines an `Interfaces` key⁸ whose value is a list of syntactically valid D-Bus interface names⁹. Each interface name may represent either a D-Bus interface, or any other “API contract”; there is no requirement that D-Bus is actually used.

⁴https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/points_of_interest/

⁵https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/points_of_interest/

⁶<https://git.gnome.org/browse/gnome-shell/tree/js/ui/remoteSearch.js>

⁷<http://standards.freedesktop.org/desktop-entry-spec/latest/>

⁸<http://standards.freedesktop.org/desktop-entry-spec/desktop-entry-spec-latest.html#interfaces>

⁹<http://dbus.freedesktop.org/doc/dbus-specification.html#message-protocol-names-interface>

67 Security considerations

68 Restricting who can advertise a given interface

69 If arbitrary ISVs¹⁰ can publish app-bundles that advertise arbitrary interfaces,
70 there is a risk that consumers of those interfaces would have an inappropriate
71 level of trust in those app-bundles by assuming that only their own app-bundles
72 can advertise “their” interfaces, for example “leaking” private information to them.

73 Communication between consumers and implementors

74 If a particular interface involves direct communication between a consumer and
75 an implementor, then discovery is not sufficient: it is also necessary to ensure
76 that the security model allows the consumer and the implementor to communi-
77 cate. Conversely, if a particular interface forces all communication between a
78 consumer and an implementor through a trusted intermediary, then it is nec-
79 essary to ensure that the security model allows both the consumer and the im-
80 plementor to communicate with the trusted intermediary, and that the trusted
81 intermediary is able to determine that forwarding data between consumer and
82 implementor will not violate the security model.

83 The desired security model for this interface is that some subset of interfaces are
84 considered to be *public interfaces*. Trusted platform components may list the
85 implementors of any interface, public or not, and may initiate communication
86 with those implementors. Store applications may list the implementors of public
87 interfaces, and may initiate communication with the implementors of public
88 interfaces, but cannot do the same for non-public interfaces.

89 Visibility of applications to other applications

90 Our security model does not consider it to be acceptable for app-bundles to
91 be able to enumerate other app-bundles’ entry points (with the exception that
92 public interfaces may be enumerated). This implies that the implementation of
93 get_implementations() (and the objects that it returns) must be done via IPC
94 (most likely D-Bus) to a trusted service, which can read the .desktop files in
95 XDG_DATA_DIRS/applications and apply appropriate filtering for the caller’s
96 limited view of the system.

97 Recommendation

98 For each entry point in a Flatpak application bundle, we recommend
99 that a freedesktop.org .desktop file is provided in the standard location
100 /app/share/applications, containing the standardized Interfaces key as
101 described above.

¹⁰https://en.wikipedia.org/wiki/Independent_software_vendor

102 This information should be made available to API users via a C API re-
 103 sembling GLib's `GAppInfo`¹¹ and `GDesktopAppInfo`¹² APIs, in particular
 104 `g_desktop_app_info_get_implementations()`¹³. However, we recommend an
 105 asynchronous version of that API in order to support the implementation being
 106 via D-Bus. Specifically, it should look something like this, with Namespace
 107 replaced by some suitable API namespace:

```

1 void namespace_app_registry_get_implementations_async (NamespaceAppRegistry *self,
2     const gchar *interface_name,
3     Gancellable *cancellable,
4     GAsyncReadyCallback *callback,
5     gpointer user_data);
6 /* Returns: (element-type GAppInfo) (transfer full): */
7 GList *namespace_app_registry_get_implementations_finish (NamespaceAppRegistry *self,
8     GAsyncResult *result,
9     GError **error);

```

108 where the result is a list of objects that implement the `GAppInfo` `GInterface`. If
 109 there is an order of preference, the most-preferred should come first. If there
 110 is no particular preference order, the implementation should use a predictable
 111 order, such as ordering by most-recently-used, most-recently-installed or alpha-
 112 betically.

113 Either this could be implemented in terms of a D-Bus API, or it could have a
 114 D-Bus API based on it for access by non-C applications, for example:

```

115 /* returns a list of pairs (desktop file ID, text of .desktop file) */
116 org.apertis.Namespace1.GetImplementations(s interface_name) → a(ss)

```

117 For interfaces (API contracts) that already have a system-wide registration
 118 mechanism, such as Telepathy connection managers, D-Bus session services and
 119 systemd user services, manual integration may be needed to ensure the registra-
 120 tion system is Flatpak-compatible.

121 Selecting a preferred implementation

122 Some of the possible use-cases for interfaces benefit from the concept of a pre-
 123 ferred implementation: for example, a navigation button should launch the pre-
 124 ferred (default) navigation application, and if points-of-interest providers have
 125 a “push” model, they should not start non-preferred navigation applications in
 126 order to push points of interest into those implementations.

127 For other use-cases, having a preferred implementation is unnecessary: for ex-
 128 ample, for a Sharing menu, global search, or points-of-interest providers with a

¹¹<https://developer.gnome.org/gio/stable/GAppInfo.html>

¹²<https://developer.gnome.org/gio/stable/gio-Desktop-file-based-GAppInfo.html>

¹³<https://developer.gnome.org/gio/stable/gio-Desktop-file-based-GAppInfo.html#g-desktop-app-info-get-implementations>

129 “pull”model, the natural design is to query all known implementations in parallel,
130 possibly excluding some that have been disabled.

131 We recommend addressing the question of a default/preferred implementation
132 on a case-by-case basis (for example by introducing a platform setting for each
133 interface that needs a preferred choice), and only developing a more general
134 solution if experience demonstrates that it is needed in practice.

135 For example, a preferred navigation application could be selected with an API
136 like

```
1 void namespace_app_registry_get_default_navigation_implementation_async (NamespaceAppRegistry *self,  
2     Gancellable *cancellable,  
3     GAsyncReadyCallback *callback,  
4     gpointer user_data);  
5 GAppInfo *namespace_app_registry_get_default_navigation_implementation_finish (NamespaceAppRegistry *self,  
6     GAsyncResult *result,  
7     GError **error);
```

137 if required.

138 The storage of preferred implementations should be considered to be an imple-
139 mentation detail of the platform component that implements this platform API.
140 For example, it could have a GSetting for each well-known interface, whose value
141 is the string app-ID (D-Bus well-known name) of the preferred entry-point, or
142 an ordered list of preferred entry-points with the most-preferred first.

143 **Enabling/disabling providers**

144 If a provider is disabled system-wide, the platform component that implements
145 interface discovery must behave as though it was not installed at all when an-
146 swering queries from other components. The storage of enabled/disabled imple-
147 mentations can be considered to be an implementation detail of that component:
148 for example it could store a string-list of disabled app IDs in GSettings. Unin-
149 stalling a provider should probably remove it from that list, so that reinstalling
150 the provider automatically enables it.

151 If a provider is disabled for a particular consumer, we recommend that the con-
152 sumer stores its own string-list of disabled app IDs, and filters the results of
153 queries on the client-side, encapsulated in a library. This probably only makes
154 sense for interfaces where the consumer will use all non-disabled implemen-
155 tations.

156 **Restricting who can advertise a given interface**

157 We recommend that interfaces advertised by a provider should be restricted by
158 app-store curators, as follows:

- each ISV¹⁴ that will publish apps on the app-store registers one or more reversed-DNS prefixes with the app-store curator as part of their app-developer account (for example, Collabora Ltd.¹⁵ might register com.collabora and/or uk.co.collabora)
- the app-store curator verifies the ISV's ownership of the relevant domain names before accepting uploads from that ISV
- app-bundles published by the ISV may implement interface names in the namespace of those reversed-DNS prefixes without necessarily triggering extensive checking by the app-store curator (for example, Collabora Ltd. could publish an app-bundle implementing com.collabora.MyInterface)
- a whitelist of known-“safe”interface names in shared namespaces such as org.apertis, org.freedesktop and org.gnome could also be implemented without necessarily triggering extra checks by the app-store curator (for example, an org.apertis.SharingProvider interface which adds the app to the Sharing menu might be considered to be “safe”for anyone to implement)
- all other interface names would be “red flags”leading to rejection or additional checking by the app-store curator

This implies that cooperating ISVs cannot invent their own interfaces without app-store curators' involvement.

It is important to note that if the platform initially has this policy, it cannot be relaxed to “anyone may implement any interface”later. If it was, ISVs writing previously-correct code would potentially become susceptible to cross-app resource access attacks (for example, if the ISV owning example.net had assumed that every implementation of net.example.MyInterface was necessarily trusted code).

Communication between consumers and implementors

App-bundles that implement public interfaces should ensure that their Flatpak permissions allow them to receive D-Bus messages from anywhere on the system.

App-bundles that do not implement public interfaces should ensure that their Flatpak permissions do not expose any services that would allow D-Bus messages from anywhere outside the application, other than any needed platform services.

Visibility of applications to other applications

App-bundles'Flatpak permissions should not include unrestricted read access to any part of the filesystem that could list other app bundles or their information, such as ~/.var or the Flatpak repository.

The implementation of the interface discovery should be done via D-Bus. The service providing this D-Bus API should be a platform component. It is con-

¹⁴https://en.wikipedia.org/wiki/Independent_software_vendor

¹⁵<https://www.collabora.com/>

197 sidered to be a trusted component for the purposes of security between app-
198 bundles: it must reveal public interface implementations to other app-bundles,
199 but must only reveal non-public interface implementations to trusted platform
200 components.