



Interface discovery

1 **Contents**

2 Use cases 2
3 In other systems 3
4 Security considerations 3
5 Restricting who can advertise a given interface 3
6 Communication between consumers and implementors 3
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*.

17 **Use cases**

- 18 • A [global search user interface](#)¹ requires a list of agents 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
39 list all the possible navigation apps.

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

²<https://em.pages.apertis.org/apertis-website/concepts/sharing/>

³https://em.pages.apertis.org/apertis-website/concepts/points_of_interest/

- 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.

54 In other systems

55 [GNOME Shell’s search provider API](#)⁶ 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).

61 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 key](#)⁸ whose value is a list of syntactically valid [D-Bus interface names](#)⁹. 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.

67 Security considerations

68 Restricting who can advertise a given interface

69 If arbitrary [ISVs](#)¹⁰ can publish app-bundles that advertise arbitrary interfaces, there is a risk that consumers of those interfaces would have an inappropriate level of trust in those app-bundles by assuming that only their own app-bundles

⁴https://em.pages.apertis.org/apertis-website/concepts/points_of_interest/

⁵https://em.pages.apertis.org/apertis-website/concepts/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>

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

72 can advertise “their” interfaces, for example “leaking” private information to
73 them.

74 **Communication between consumers and implementors**

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

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

90 **Visibility of applications to other applications**

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

98 **Recommendation**

99 For each application or agent (*entry point*) in an application bundle, we rec-
100 ommend that a freedesktop.org `.desktop` file is provided in a standard location
101 such as `/var/lib/apertis_extensions/applications` by installing the application
102 bundle. Possible implementations of this:

- 103 • The store publication process could verify that the contents of the provided
104 `.desktop` file are appropriate for the application’s manifest.
- 105 • The store publication process could generate a `.desktop` file from the appli-
106 cation’s manifest, with no control from the application author, other than
107 to the extent that they can control the manifest and still have it approved
108 by the app-store curator.
- 109 • The application manager could generate a `.desktop` file from the applica-
110 tion’s manifest during installation.

111 The resulting `.desktop` file should contain the standardized `Interfaces` key as
112 described above.

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

```
119 void namespace_app_registry_get_implementations_async (NamespaceAppReg-  
120 istry *self, const gchar *interface_name, Gancellable *can-  
121 cellable, GAsyncReadyCallback *callback, gpointer user_data);  
122 /* Returns: (element-type GAppInfo) (transfer full): */ GList *names-  
123 pace_app_registry_get_implementations_finish (NamespaceAppRegistry *self,  
124 GAsyncResult *result, GError **error);
```

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

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

```
132 /* returns a list of pairs (desktop file ID, text of .desktop file) */  
133 org.apertis.Namespace1.GetImplementations(s interface_name) -> a(ss)
```

134 For interfaces (API contracts) that already have a system-wide registration
135 mechanism, such as Telepathy connection managers, D-Bus session services
136 and systemd user services, we recommend adopting the existing mechanism
137 instead, using appropriate subdirectories of `/var/lib/apertis_extensions` where
138 necessary.

139 **Selecting a preferred implementation**

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

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

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>

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

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

153 For example, a preferred navigation application could be selected with an API
154 like

```
155 void namespace_app_registry_get_default_navigation_implementation_async (Names-  
156 paceAppRegistry *self,          Gancellable *cancellable,          GAsyn-  
157 cReadyCallback *callback,      gpointer user_data); GAppInfo *names-  
158 pace_app_registry_get_default_navigation_implementation_finish (NamespaceAp-  
159 pRegistry *self,      GAsyncResult *result,      GError **error);
```

160 if required.

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

166 **Enabling/disabling providers**

167 If a provider is disabled system-wide, the platform component that implements
168 interface discovery (for example Didcot) must behave as though it was not
169 installed at all when answering queries from other components. The storage of
170 enabled/disabled implementations can be considered to be an implementation
171 detail of that component: for example it could store a string-list of disabled app
172 IDs in GSettings. Uninstalling a provider should probably remove it from that
173 list, so that reinstalling the provider automatically enables it.

174 If a provider is disabled for a particular consumer, we recommend that the con-
175 sumer stores its own string-list of disabled app IDs, and filters the results of
176 queries on the client-side, encapsulated in a library. This probably only makes
177 sense for interfaces where the consumer will use all non-disabled implementa-
178 tions.

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

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

- 182 • each ISV¹⁴ that will publish apps on the app-store registers one or
183 more reversed-DNS prefixes with the app-store curator as part of their

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

184 app-developer account (for example, [Collabora Ltd.](https://www.collabora.com/)¹⁵ might register
185 `com.collabora` and/or `uk.co.collabora`)

- 186 • the app-store curator verifies the ISV's ownership of the relevant domain
187 names before accepting uploads from that ISV
- 188 • app-bundles published by the ISV may implement interface names in the
189 namespace of those reversed-DNS prefixes without necessarily triggering
190 extensive checking by the app-store curator (for example, Collabora Ltd.
191 could publish an app-bundle implementing `com.collabora.MyInterface`)
- 192 • a whitelist of known-“safe” interface names in shared namespaces such
193 as `org.apertis`, `org.freedesktop` and `org.gnome` could also be implemented
194 without necessarily triggering extra checks by the app-store curator (for
195 example, an `org.apertis.SharingProvider` interface which adds the app to
196 the Sharing menu might be considered to be “safe” for anyone to imple-
197 ment)
- 198 • all other interface names would be “red flags” leading to rejection or ad-
199 ditional checking by the app-store curator

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

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

208 **Communication between consumers and implementors**

209 App-bundles that implement of public interfaces should receive AppArmor pro-
210 files allowing them to receive D-Bus method calls from anywhere.

211 App-bundles that do not implement public interfaces should receive AppArmor
212 profiles that allow D-Bus method calls from platform services, and from other
213 processes from the same app-bundle, but deny other D-Bus method calls.

214 **Visibility of applications to other applications**

215 App-bundles' AppArmor profiles should not give them read access to
216 `/var/lib/apertis_extensions/applications` or to other app-bundles' manifests.

217 The implementation of the interface discovery should be done via D-Bus. The
218 service providing this D-Bus API (for example `Didcot`) should be a platform
219 component. It is considered to be a trusted component for the purposes of
220 security between app-bundles: it must reveal public interface implementations
221 to other app-bundles, but must only reveal non-public interface implementations
222 to trusted platform components.

¹⁵<https://www.collabora.com/>