Content hand-over

Contents

Content handover is when an application is asked by the user to open a file it doesn't know how to handle or to start a service it is not responsible for providing. For example, a browser just finished downloading an epub file and the user now wants to open it. The browser will ask the system to start the application responsible for dealing with that file. Another example would be a map application in which the user located a destination and now wants to navigate. The map app will ask the system to start the navigation app for that location.

Use-cases

See Content hand-over/Use-cases[1].

The API

The SDK will have a very simple API for applications to hand-over:

```
void launch_app_for_uri(const char* uri)
```

This API will make a D-Bus call to the launcher service, Didcot. See below.

The Manifest and schemes

An application that is able to handle a given content type should declared that in the manifest. Collabora recommends a section in the manifest for supported content types containing a list of those supported. A different section should contain the schemes the app can handle. Collabora recommends the schemes used by iOS[2] should be adopted as much as possible for consistency and to take advantage of existing knowledge.

[1] https://em.pages.apertis.org/apertis-website/concepts/content_hand-over_use-cases/

[2] https://developer.apple.com/library/ios/featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html

When more than one app can handle a scheme or content type

Since content types and schemes can be handled by more than one application, Didcot should be capable of providing the user with a list of options. It can be similar to Android, in which the user is able to tell the system to always choose a given application for a specific type or scheme.

The App Launcher can handle specific schemes or content types specially. For instance, a scheme for getting information about a Point of Interest (POI) can have a special dialog that besides showing the applications that can provide details on the POI also shows a number of details itself.

Arguments

Any information that needs to be passed to the application should be encoded in the URI. For instance, a navigation app may want an address to navigate to. It might be encoded like this:

```
nav:30+Destination+Road?saddr=21+Source+Street
```

A scheme can be provided to get more information about a song. Details about Shakira's Sale el Sol song could be requested like this:

```
media-info:Sale+el+Sol?author=Shakira&album=Waka+Waka
```

The arguments can be flexible, as well. The requester might be able to provide information which are more or less specific depending on its own context. So a query for information on a location for instance can be either a very specific latitude, longitude:

```
location-info:Grömitz?lat=54.174730&long=10.977516
```

A detailed description for a place in a city:

```
location-info:Carat Hotel?type=hotel&city=Grömitz&country=Germany
```

Or even a number of stops on the way to a destination:

```
location-info:Grömitz?lat=54.174730&long=10.977516&stop=Bochum&lat=xx&long=yy&stop=Berlin&lat=zz&long=ww
```

The allowed arguments, their formats and meanings should be part of the design for the specific service, and should be properly documented. Convenience APIs can and should be provided for services, to make them easier to use.

The service: Didcot

Didcot is the service which handles launching applications. It will be responsible for taking requests from applications for handling files or schemes. The service will rely on the manifest information provided by the applications to know which application knows how to handle a given file format or scheme. It will provide a single method:

```
<method name="LaunchAppForURI">
<annotation name="org.freedesktop.DBus.GLib.Async" value=""/>
<arg name="uri" direction="in" type="s"/>
</method>
```

When given file:// URIs, Didcot will sniff the content type from the file and look for applications that can handle that type on the manifest database (or its own cached version of it). Similarly, when given any non-file URI, Didcot will look for apps that can handle that scheme. If there is more than one application that can handle the file or scheme, Didcot may decide to present a dialog asking the user to select which one they would prefer to use.

Didcot can treat certain schemes or content types as privileged and allow only certain apps to handle them.

A note about content type and scheme ambiguity

The only situation in which Didcot should use content types is for local files. A mail attachment that needs to be opened should be saved to local storage and then opened like a regular file, as is done in other mobile systems such as Android. Schemes such as http may host a great number of file types. Only browsers should register for that scheme, though. If the browser cannot handle a specific file format, it will download the file to local storage and open it using a file:// URI. It is tempting to try to be smarter and automatically decide that a file hosted on a web site should actually be handled by a music or video player, for instance. That does not work well in practice: web servers lie about file types and file names lie even more. Even browsers, which are very well equipped to work with that fact are sometimes fooled. The best approach is to let browsers deal with web servers and let Didcot handle local files on the apps' behalf.

See also

- Content hand-over/Design issues[3].

[3] https://em.pages.apertis.org/apertis-website/concepts/content_hand-over_design_issues/