



Content hand-over Design issues

1 **Contents**

2 **Design issues for content hand-over** **2**

3 Creating new URI schemes (or not) 2

4 D-Bus or not? 2

5 Can we use GDesktopAppInfo rather than reinventing it? 3

6 Should Didcot and Canterbury be the same thing? 3

7 Maintaining established communications 3

8 Registration mechanism 3

9 Security considerations 4

10 URI scheme/media-type hijacking 4

11 Bidirectional content handover 4

12 Terminating the launched app 5

13 **Design issues for content hand-over**

14 **Creating new URI schemes (or not)**

15 The W3C suggests [avoiding creation of new URI schemes](#)¹ but instead dispatch-
 16 ing instead on the type of the representation (i.e. the content-type). Creating
 17 new content-types is [substantially easier than it used to be](#)² and is compatible
 18 with sending those content-types via http or email.

19 [data:application/vnd.myapp.myobject,here-is-my-object](#) is one way to pass
 20 small bits of content-typed data around as URIs, but we should be careful not
 21 to overuse inline data.

22 Ascribing semantics to a subtree of http, like Apple does for [[https://developer.
 23 apple.com/library/ios/featuredarticles/iPhoneURLScheme_Reference/
 24 MapLinks/MapLinks.html#//apple_ref/doc/uid/TP40007899-CH5-SW1](https://developer.apple.com/library/ios/featuredarticles/iPhoneURLScheme_Reference/MapLinks/MapLinks.html#//apple_ref/doc/uid/TP40007899-CH5-SW1)”
 25 map links] in recent iOS, is another way to avoid new URI schemes.

26 **D-Bus or not?**

27 D-Bus has some subtleties for how it interacts with AppArmor: the caller of a
 28 method, or the sender of a signal, needs permission to send to the recipient (in
 29 its profile), and the recipient needs permission to receive from the sender (in
 30 *its* profile). We are trying to avoid relying on fine-grained message filtering
 31 (e.g. by interface) because that won't work on kdbus. This means we might
 32 have to work out something clever here if we want pairs of arbitrary apps from
 33 different app-bundles to be able to interact directly: we'd need either some way
 34 to have “equivalence classes” of apps, or a trusted D-Bus proxy that rate-limits
 35 and filters messages.

¹<http://www.w3.org/TR/webarch/#URI-scheme>

²<https://www.iana.org/form/media-types>

36 Another possibility would be for Didcot (or Canterbury) to proxy autho-
37 rized requests between app-bundles; we could have the requester call a
38 method on Didcot that results in Didcot calling `o.fd.Application.Open` or
39 `o.fd.Application.Activate`³ on the provider, if it considers the provider to be
40 suitable.

41 This is fine for “launch” and “open URI”, but not really up to the job for a
42 more complex interface (search-provider-style) or for an interface with more
43 data (search results coming back). We could potentially have an API through
44 which we fd-pass a socket, or pass through an abstract socket by name, or
45 something, and then do D-Bus over that; but then we lose total ordering of
46 messages, and become sad.

47 In the non-D-Bus corner, we could use a mesh of 1-1 connections between apps;
48 but then we have a mesh of 1-1 connections between apps, we’ve still lost total
49 ordering, and we potentially need to reinvent message framing.

50 smcv’s instinct here is to use D-Bus for everything that is a one-off action in
51 response to something the user does; seriously consider using D-Bus for query-
52 style APIs; and probably avoid D-Bus for TPEG, since that presumably already
53 has framing, and we might end up doing the filtering navigation-app-side.

54 **Can we use GDesktopAppInfo rather than reinventing it?**

55 GAppInfo provides nice APIs for the basics of what Didcot does.

56 Unfortunately, it assumes that applications can be launched with direct D-Bus
57 activation (not true if we are relying on Didcot for launching, unless we have
58 additional infrastructure that helps us out) and that they have .desktop files.

59 Options:

- 60 • patch GDesktopAppInfo for Apertis
- 61 • write our own high-level API which will end up rather similar (potentially
62 even identical), perhaps in its own LGPL-licensed library so it can copy
63 bits from GDesktopAppInfo, and optionally patch GLib documentation
64 to say not to use GDesktopAppInfo
- 65 • require that every app/agent is DBusActivatable, generate .desktop files
66 from their manifests, and use a D-Bus proxy to transform activation re-
67 quests into what we need
- 68 • generate .service files so that every app/agent **is** DBusActivatable, use
69 a D-Bus proxy or clever AppArmor rules to make that work, and bypass
70 Didcot entirely

71 **Should Didcot and Canterbury be the same thing?**

72 Canterbury is a trusted intermediary for launching apps. So is Didcot, if you
73 think about it. Maybe they should merge?

³<http://standards.freedesktop.org/desktop-entry-spec/1.1/ar01s08.html>

74 Maintaining established communications

75 Do we have a use-case for this, that is not already satisfied by “list providers
76 and start querying them again”?

77 Registration mechanism

78 It looks like we’re going to need:

- 79 • I handle foo: (scheme)
- 80 • I handle everything in foo://bar (scheme + authority)
- 81 • I handle everything starting with foo://bar/baz/ (scheme + authority
82 – path-prefix) (?)
- 83 • I open files of type foo/bar
- 84 • I open files of type foo/* (?)
- 85 • I open all files (?)
- 86 • I can share files of type foo/bar, foo/* (?), all files

87 and for the streaming-ish use cases:

- 88 • I implement org.apertis.PointOfInterestProvider (etc.)

89 This is, not coincidentally, a lot like .desktop files. We could make the encoding
90 in the manifest somewhat similar, and auto-generate a .desktop file if desired.
91 Something like this (using YAML here because it’s easier to hand-write, apply
92 the obvious mapping into JSON for production):

```
93 entry_points:      com.example.MyApp:      content_types:      -  
94 foo/bar           - foo/*           url_handler:        - "foo:"           -  
95 "foo://bar"       - "foo://bar/baz/"  com.example.MyAgent: implements:  
96 - org.apertis.PointOfInterestProvider
```

97 Security considerations

98 URI scheme/media-type hijacking

99 It is important to note that URI schemes and media types will, in general,
100 be a “first come, first served” shared resource. The Scheme Hijacking attack
101 described in [Unauthorized Cross-App Resource Access on MAC OS X and iOS](#)⁴
102 §3.4 relies on the attacker registering for a URI scheme that an app developer
103 had (mis)used as a general IPC mechanism.

104 One thing we can do to improve on iOS’ behaviour here is to provide IPC
105 mechanisms that automatically convey app and user identity information that
106 cannot be faked, such as D-Bus, and document them as a better way to solve
107 the problems that iOS app developers are trying to solve by making up URI
108 schemes.

⁴<https://drive.google.com/file/d/0BxxXk1d3yyuZOFIsdkNMSGswSGs/view?pli=1>

109 For example, there should be a way for a pair of cooperating applications to
110 declare in their app-bundle manifests that each may communicate with the other
111 via D-Bus. Executables in the same app-bundle should just be able to do D-Bus
112 to each other anyway, no questions asked.

113 Documentation for the content handover feature should recommend these other
114 IPC mechanisms, and caution against using content handover to transfer cre-
115 dentials.

116 **Bidirectional content handover**

117 One feature that was considered is bidirectional content handover. We recom-
118 mend treating this as out-of-scope: it requires thought to be put into security
119 between apps, and in particular what we want to allow apps to do. If general
120 bidirectional channels between pairs of apps are required, they should use a
121 protocol such as D-Bus or an AF_UNIX socket, which provides secure authen-
122 tication (credentials that cannot be faked, including the uid and AppArmor
123 profile). ()

124 **Terminating the launched app**

125 Suppose app A launches app B via content handover. One design question that
126 was considered was whether app A should be able to terminate app B.

127 We recommend that this capability is not offered: if it was misused, it would be
128 easy for a user to misinterpret it as app B crashing. ()