AllJoyn Introspection XML Proposal

# Executive Summary

Currently there are 3 incompatible introspection XML formats:

1. D-Bus introspection XML, as returned from Introspect(), and defined in <http://dbus.freedesktop.org/doc/dbus-specification.html#introspection-format>.
2. Extended introspection XML, which isn’t returned by anything, but which is defined in <https://wiki.allseenalliance.org/irb/extended_introspection_xml>.  This uses named structs in type signatures, and so is incompatible with D-Bus introspection XML since any code expecting D-Bus introspection XML will treat named structs in type signatures as illegal.
3. AllJoyn introspection XML, as returned from IntrospectWithDescription().  Its format is not documented anywhere but in short: it uses D-Bus style type signatures as does (1), but uses the <description> element and signal behavior attributes from (2). As such, it is treated as legal by most tools using D-Bus introspection XML, except ones that do strict validation on elements and attributes.

This document proposes collapsing them and only having one format going forward, which is D-Bus introspection compatible, in order to maximize use of existing tools expertise across the industry.

# Scenarios and Requirements

There are multiple scenarios that need to be (or already are, at least in part) solved:

1. End users need descriptions of interfaces and their members, for creating If-This-Then-That (IFTTT) style rules as discussed in <https://wiki.allseenalliance.org/_media/compliance/alljoyn_events_actions_feature_interface_definition.pdf>
2. Administrators need descriptions of interfaces and sometimes their members, for doing ACLs (see the discussion of descriptions under “Application Manifest” in <https://allseenalliance.org/developers/learn/core/security2_0/hld>).  It isn’t stated that the description is obtained in introspection XML format, so that may or may not be the case, because it hasn’t been defined yet.  However, the advantage of using introspection XML is that it avoids gratuitously introducing yet a fourthvariation of an interface description format.
3. The IRB needs to review interface definitions.
4. A developer wants to get the official XML for a standard interface, from the Alliance website, for code generation purposes.  In one sense, this is dangerous though: in AllJoyn, the code, not the documentation, is authoritative.  Example: The published Onboarding interface spec is currently wrong and anyone who uses it will fail to interoperate with any device. Specifically, <https://allseenalliance.org/developers/learn/base-services/onboarding/interface> says a method is called “ConfigWifi” but the introspection XML section of the same document calls it “ConfigureWifi”, and unfortunately *both* are wrong.  The code has “ConfigureWiFi” (note the capital F) and the protocol is case-sensitive.   That can be partially mitigated by good IRB review, and by getting ASADOC bugs fixed in a timely fashion, which doesn’t happen today.
5. A developer wants to get the actual XML from a device (which may or may not be a proprietary interface) for code generation purposes.   You cannot guarantee the XML will be available anywhere else, and even if it were, you cannot guarantee anything else will interoperate with a device (see the Onboarding existence proof above).

The corollary is that the introspection XML returned by a device at runtime must be complete; it cannot be a subset of the formally published XML on [www.allseenalliance.org](http://www.allseenalliance.org) for any standard interface supported by the device. If it deviates from that, it must be as a superset (e.g., because of vendor-specific annotations that were added as discussed in section 4.1).

## Code Generators

Code generators (of which at least two AllJoyn-specific ones are currently known to exist, and some discussion at <https://developer.gnome.org/gio/stable/gdbus-codegen.html> for D-Bus is also relevant) need to support both scenario 4 and scenario 5. Today code generation is complicated by the fact that the extended introspection XML format is designed to make code generation easy, but it is not available in scenario 5 today, and for scenario 4, no published interfaces yet use the extended format. When ones do get published, code generators would need to support multiple incompatible formats.

The limitation of the formats used today is that structures and members cannot be named, complicating code generation significantly, and making generated code hard to read and maintain.

As such, the requirements are:

1. Use a single common format, in order to reduce complexity
2. Retain D-Bus compatibility, in order to leverage existing tools and expertise
3. Make code generation straightforward

# Interface XML Language

## Versioning

Use “org.gtk.GDBus.Since” (as defined in <http://dbus.freedesktop.org/doc/dbus-api-design.html#annotations> and further discussed at <https://developer.gnome.org/gio/stable/gdbus-codegen.html>) for versioning and schema evolution.

This annotation can be used on any <interface> element to specify the version of the interface. This annotation can be used on any <method>, <signal> and <property> element to specify the interface version the member first appeared in. For AllJoyn, the value must be an integer, and each member version must be less than or equal to the interface version. This annotation can be omitted from all elements in the initial version of an interface, and the default value if omitted is “1”.

Example:

<interface name="org.alljoyn.example.Test">  
  <annotation name="org.alljoyn.Interface.Version" value="2"/>

  <!-- Since no annotation exists below, Foo was in version 1. -->  
  <property name="Foo" type="s" access="read"/>

  <!-- Bar was also in version 1, this illustrates that it's not an  
       error to have version="1" be explicit, but it's unnecessary. -->  
  <property name="Bar" type="s" access="read">  
    <annotation name="org.alljoyn.Interface.Version" value="1"/>  
  </property>

  <!—Baz was added in version 2. -->  
  <property name="Baz" type="s" access="read">  
    <annotation name="org.alljoyn.Interface.Version" value="2"/>  
  </property>  
</interface>

## Type Signatures

Here is an example of the current Extended Introspection XML format using named types:

<struct name="ObjectDescription">  
  <field name="path" type="o"/>  
  <field name="implementedInterfaces" type="as"/>  
</struct>

<method name="GetObjectDescription">  
  <arg name="objectDescription" type="a[ObjectDescription]" direction="out"/>  
</method>

<signal name="Announce" sessionless="true">  
  <arg name="objectDescription" type="a[ObjectDescription]"/>  
  <arg name="metadata" type="a{sv}"/>  
</signal>

As noted earlier, the above type signatures are incompatible with the D-Bus introspection XML format. The proposed D-Bus compatible equivalent using annotations would be:

<annotation name="org.alljoyn.Bus.Struct.ObjectDescription.Field.path.Type"  
 value="o"/>  
<annotation name="org.alljoyn.Bus.Struct.ObjectDescription.Field.  
implementedInterfaces.Type"  
 value="as"/>

<method name="GetObjectDescription">  
  <arg name="objectDescription" type="a(sas)" direction="out"/>  
 <!-- In D-Bus, methods can have annotations, but args cannot,  
 so an arg annotation must be on the method. -->   
  <annotation name="org.alljoyn.Bus.Arg.objectDescription.Type”  
 value="a[ObjectDescription]”/>  
</method>

<signal name="Announce">  
  <arg name="objectDescription" type="a(sas)"/>  
 <!-- In D-Bus, signals can have annotations, but args cannot,  
 so an arg annotation must be on the signal. -->  
  <annotation name="org.alljoyn.Bus.Arg.objectDescription.Type"  
 value="a[ObjectDescription]"/>

  <arg name="metaData" type="a{sv}"/>  
</signal>

That is, the following annotation conventions are defined:

* **org.alljoyn.Bus.Struct.*StructureName*.Field.*fieldName*.Type:** This defines a field named *fieldName* in a structure named *StructureName*, with a type signature given in the value. The value has the same syntax as currently allowed by Extended Introspection XML types.
* **org.alljoyn.Bus.Arg.*argName*.Type:** This defines the type, using the same syntax as currently allowed by Extended Introspection XML types, of a signal or method argument named *argName*.

## Descriptions and Comments

All five scenarios need to obtain some descriptions of interfaces and members, whether to display to an end-user or admin, or simply to generate comments in generated code. However, there are two different audiences, with different requirements:

1. End users/administrators (scenarios 1-2) where having descriptions localized into different languages is important, but explanations of low-level implementation details would be confusing.
2. Developers (scenarios 3-5) where localized descriptions is less important, since Alliance specs for example are available only in English, but explanations of low-level implementation details may be required.

The proposal is:

1. End-user accessible descriptions use XML annotations that replaces the <description> element currently defined by AllJoyn’s Extended Introspection XML. Alliance interfaces require English description elements, and descriptions in other languages are optional. For security, descriptions (regardless of language) must be verified to come from the organization defining the interface.
2. Additional *developer-only* details are not localized, and XML <!-- comment --> blocks containing gtk-doc formatted text are used as discussed at <http://dbus.freedesktop.org/doc/dbus-api-design.html#documentation>. End-user accessible descriptions should not be duplicated in such comments. That is, developers get both types and comment blocks should only be used for additional details not obvious from the end-user description.
3. IntrospectWithDescription() currently returns XML that is not strictly D-Bus compliant. It should continue to do so for backwards compatibility, but should be deprecated.
4. Introspect() should be updated to include all end-user-accessible descriptions as well as developer-only comments.

For developer-only descriptions, the org.gtk.GDBus.DocString XML annotation is also defined at <http://dbus.freedesktop.org/doc/dbus-api-design.html#documentation> which explains that <!—comment --> blocks are preferred. As such, AllJoyn need not support org.gtk.GDBus.DocString XML annotations. Furthermore, that annotation does not support language tags, and hence is not suitable for end-user accessible descriptions either.

For end-user descriptions, the annotation org.alljoyn.Bus.DocString.*LanguageTag* is hereby defined, where *LanguageTag*, is a language tag with any hyphens converted to underscores (as is done with the DNS name prefix). Example:

<signal name="LightOn">  
 <annotation name="org.alljoyn.Bus.DocString.En"  
 value="Light has been turned on"/>  
 <annotation name="org.alljoyn.Bus.DocString.Nl"  
 value="Licht is aangestoken"/>  
</signal>

The security 2.0 subgroup had a long description about interface descriptions for scenario 2 (application manifests), and whether those descriptions could come from a device or whether they could only come from a server in the cloud, keeping in mind that sometimes one may not have connectivity to the Internet.  They tentatively concluded that they could only safely come from the cloud, such as from a site maintained by the org identified in the interface prefix.   These same arguments might also apply to scenario 1 (events & actions) descriptions.

## Signal Emission Behaviors

The introspection XML format used by IntrospectWithDescription() currently allows specifying the signal emission behavior via XML attributes on the <signal> element. These are not strictly compliant with the D-Bus introspection XML format. As such, the following annotations are defined for D-Bus compliance, for use in <signal> elements.

| **Name** | **Values (separated by ,)** | **Description** |
| --- | --- | --- |
| org.alljoyn.Bus.Signal.Sessionless | true,false | Whether or not the signal may be sessionless. Equivalent to the existing “sessionless” attribute. |
| org.alljoyn.Bus.Signal.Sessioncast | true,false | Whether or not the signal may be sessioncast. Equivalent to the existing “sessioncast” attribute. |
| org.alljoyn.Bus.Signal.Unicast | true,false | Whether or not the signal may be unicast. Equivalent to the existing “unicast” attribute. |
| org.alljoyn.Bus.Signal.GlobalBroadcast | true,false | Whether or not the signal may be global broadcast. Equivalent to the existing “globalbroadcast” attribute. |

All signal emission behavior annotations have a default value of false if any signal emission behavior annotation are present, and a default value of true if no such annotations are present.

## Use of Annotations vs Const Properties

Constants such as the units, minimum value, and maximum value of a given property are important to be able to communicate to a developer and consume in code (e.g., for a UI) or use during code generation time.

The following AllJoyn-specific annotations are defined:

* org.alljoyn.Bus.Property.Min: Value is a number expressed as a string
* org.alljoyn.Bus.Property.Max: Value is a number expressed as a string
* org.alljoyn.Bus.Property.Units: Value is a string suitable for display

If the constant is defined by the interface *and it must be the same for every device and implementation of that interface*, then use an annotation. This is because annotations are not per-device and can be cached and applied to any device. Example:

<property name="Temperature" type="u" access="readwrite">  
 <annotation name="org.alljoyn.Bus.Property.Max" value="100"/>  
 ...  
</property>

If the constant can vary by implementation or by device, then use a const property, since these are cached on a per-device basis. Example:

<property name="Temperature" type="u" access="readwrite">  
 ...  
</property>  
<property name="MaxTemperature" type="u" access="read">  
 <annotation name="org.freedesktop.DBus.Property.EmitsChangedSignal"  
 value="const"/>  
 ...  
</property>

## Custom Annotations

Only D-Bus and AllJoyn standard annotations may appear in “official” XMLs that are published by the AllSeen Alliance. Vendors free to add their own annotations in introspection XML (including XML for AllJoyn standard interfaces) returned at runtime by devices, or in XMLs posted on their own servers.

Any tool that parses XML must be able to copy with unknown annotations anyway, for reasons of forward compatibility.

Custom annotation names must follow the same guidelines as for interface names and error names.