How managed is your D-Bus?

Alp Toker
What is D-Bus?

D-Bus is an inter-process communication framework that lets applications interface with the system event bus as well as allowing them to talk to one another in a peer-to-peer configuration.
What is managed D-Bus?

• A clean-room MIT/X11 implementation of the D-Bus protocol in C# providing full integration with the CLR/.NET type system

• Runs on the Mono and Microsoft Common Language Runtime platforms

• Provides seamless, high-performance access to/from D-Bus using any of several programming languages including C#, IronPython, Boo, VB, Nemerle...
And..?

- Used by over a dozen GNOME applications – everything from single instance detection to complex instant messaging tasks and object componentry
- Managed D-Bus was the first independent implementation of D-Bus, and has encouraged other platform developers who wish to avoid the reference implementation
- Aims to provide a compelling alternative to libdbus
Managed D-Bus features

- Direct mapping to/from C# interfaces and types
- Support for D-Bus structs using C# structures
- Support for typed D-Bus dictionaries using C# generics
- Mapping to/from D-Bus signals to C# events
- Mapping to/from D-Bus errors/CLR exceptions
- Good performance thanks to dynamically compiled optimised marshalers
- Full thread-safety and concurrent messaging support
- Portable: truly build-once run-anywhere
Licensing: Getting it right

libdbus, reference implementation
GPL + AFL = loss

dbus-java, ruby-dbus
LGPL, very reasonable

Managed D-Bus
MIT X11 = win

Please put me out of my misery
GLib integration

• Managed D-Bus encourages single-threaded asynchronous messaging where possible
• ndesk-dbus-glib is a simple package that integrates ndesk-dbus with the GLib main loop
• Has its own release cycle and integration with other main loops is possible without modification of the core managed D-Bus library
• Popular with Gtk# applications
• Should not be used in headless apps
Example: Hooking up to HAL

- HAL has a notoriously bad D-Bus API
- We will use it as an example anyway
Defining a D-Bus interface: Hal.Device

/* an Interface attribute is applied to mark the D-Bus interface name */
[Interface ("org.freedesktop.Hal.Device")]
public interface Device
{
    /* event definitions map to D-Bus signals */
    event PropertyModifiedHandler PropertyModified;

    /* properties and method calls map to D-Bus calls */
    /* the D-Bus type system is capable of representing generic dictionaries */
    IDictionary<string, object> AllProperties { get; }

    /* further methods and signals omitted for brevity */
}

In this example we are only going to import the interface, but exporting it would simply be a matter of implementing the interface in any given class and Register()ing it with the bus.
...and necessary structures and delegates

/* D-Bus supports arbitrary structures */
public struct PropertyModification
{
    public string Key;
    public bool Added;
    public bool Removed;
}

public delegate void PropertyModifiedHandler (int modificationsLength, PropertyModification[] modifications);
Bringing it all together: Using HAL

/* request a proxy object for the device manager from the system message bus */
       new ObjectPath("/org/freedesktop/Hal/Manager"));

/* enumerate all devices */
foreach (Device dev in mgr.AllDevices) {
    /* print the path of the object */
    Console.WriteLine (dev.ToString ());

    /* hook up to the PropertyModified signal of the device */
    dev.PropertyModified += delegate (int modificationsLength,
                            PropertyModification[] modifications) {
        /* when properties are modified, print the changes */
        Console.WriteLine ("Properties changed on device {0}:", dev);
        foreach (PropertyModification modification in modifications)
        Console.WriteLine (modification.Key);
    };
}
D-Bus XML Introspection

- Managed D-Bus generates XML introspection data dynamically on demand from ordinary C#/.NET interfaces

```xml
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">

<node name="/org/freedesktop/sample_object">
  <interface name="org.freedesktop.SampleInterface">
    <method name="Frobate">
      <arg name="foo" type="i" direction="in"/>
      <arg name="bar" type="s" direction="out"/>
      <arg name="baz" type="a{us}" direction="out"/>
      <annotation name="org.freedesktop.DBus.Deprecated" value="true"/>
    </method>
    <method name="Bazify">
      <arg name="bar" type="(iiu)" direction="in"/>
      <arg name="bar" type="v" direction="out"/>
    </method>
    <method name="Mogrify">
      <arg name="bar" type="(iiav)" direction="in"/>
    </method>
    <signal name="Changed">
      <arg name="new_value" type="b"/>
    </signal>
    <property name="Bar" type="y" access="readwrite"/>
  </interface>
  <node name="child_of_sample_object"/>
  <node name="another_child_of_sample_object"/>
</node>
```
D-Bus: The protocol

- Binary protocol
- Support for big/little endian messages
- Lies between raw sockets and CORBA in terms of complexity
- Less complex/featured than SOAP and WCF
- More featureful than e.g. XML RPC
<table>
<thead>
<tr>
<th>Conventional Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>121 (ASCII 'y')</td>
<td>8-bit unsigned integer</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>98 (ASCII 'b')</td>
<td>Boolean value, 0 is FALSE and 1 is TRUE. Everything else is invalid.</td>
</tr>
<tr>
<td>INT16</td>
<td>110 (ASCII 'n')</td>
<td>16-bit signed integer</td>
</tr>
<tr>
<td>UINT16</td>
<td>113 (ASCII 'q')</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>INT32</td>
<td>105 (ASCII 'i')</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>UINT32</td>
<td>117 (ASCII 't')</td>
<td>32-bit unsigned integer</td>
</tr>
<tr>
<td>INT64</td>
<td>120 (ASCII 'x')</td>
<td>64-bit signed integer</td>
</tr>
<tr>
<td>UINT64</td>
<td>116 (ASCII 't')</td>
<td>64-bit unsigned integer</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>100 (ASCII 'd')</td>
<td>IEEE 754 double</td>
</tr>
<tr>
<td>STRING</td>
<td>115 (ASCII 's')</td>
<td>UTF-8 string (must be valid UTF-8). Must be nul terminated and contain no other nul bytes.</td>
</tr>
<tr>
<td>OBJECT_PATH</td>
<td>111 (ASCII 'o')</td>
<td>Name of an object instance</td>
</tr>
<tr>
<td>SIGNATURE</td>
<td>103 (ASCII 'g')</td>
<td>A type signature</td>
</tr>
<tr>
<td>ARRAY</td>
<td>97 (ASCII 'a')</td>
<td>Array</td>
</tr>
<tr>
<td>STRUCT</td>
<td>114 (ASCII 'r'), 40 (ASCII '('), 41 (ASCII ')')</td>
<td>Struct</td>
</tr>
<tr>
<td>VARIANT</td>
<td>118 (ASCII 'v')</td>
<td>Variant type (the type of the value is part of the value itself)</td>
</tr>
<tr>
<td>DICT_ENTRY</td>
<td>101 (ASCII 'e'), 123 (ASCII '{'), 125 (ASCII '}')</td>
<td>Entry in a dict or map (array of key-value pairs)</td>
</tr>
</tbody>
</table>
Single precision floating point

- A managed D-Bus protocol extension
- Type code 'f'
- 32-bit single precision floating point
- Defined by IEEE 754
- Essential for supporting the full range of types in modern execution environments
- Now supported by dbus-java, upcoming support in dbus-ruby, dbus-python and probably other implementations
Talking D-Bus

D-Bus message

Header

Body
IronPython and the DLR

- IronPython is an Open Source implementation of Python targeting the CLR
- Predecessor to the Dynamic Language Runtime
- Managed D-Bus can support these platforms
- Can dynamically introspect remote interfaces and construct local proxies for them, much like dbus-python
Legacy-free, API/ABI stable

- Modern, clean API based around the final D-Bus terminology
- No concept of a “service” in D-Bus 1.0 – our API is free of this term
- Very few entry points, all with well-defined behaviour
- The low-level D-Bus API is internal ie. not available in the public API
Hindsight is 20/20

• Learning from previous attempts
  – Joe Shaw's original dbus-sharp binding
    • A good start, but became unmaintained
    • Memory management issues and lack of features (structures, dictionaries etc.)
  – Adam Lofts' dbus-sharp (part of the Chatter IM client)
    • Closer to a modern dbus-sharp API; Showed that you don't have to expose low-level API to have a usable binding
    • Still suffered from some memory management issues and lack of features
Distributions and packaging

• Shipping with Ubuntu Feisty as part of 'main'
• Sebastian Dröge created and continues to maintain the Debian/Ubuntu packages libndesk-dbus1.0-cil and libndesk-dbus-glib1.0-cil
• Going into SLED/OpenSUSE
• Going into Fedora
• Gentoo ebuild available
Part of the future GNOME mobile/embedded platform?

- Works well on eg. the Nokia 770/N800
- Independent of the version of libdbus installed on the device (which has been out of pace with desktop versions in the past)
- High performance thanks to the Mono ARM JIT
- Build on the desktop, run on the device: No cross-compilation
GNOME: It's all about the applications

- Infrastructure code like IPC should just work: application developers are not expected to have expertise in this area.
- Infrastructure code is easy and boring... let's take a look at some applications
Users of managed D-Bus: Tomboy

Uses D-Bus for single instance detection, shell remote control and more recently for note synchronisation with Conduit
Users of managed D-Bus: F-Spot

Uses D-Bus for single instance detection, shell remote control. There is current interest in exposing more of F-Spot's database to the desktop.
Users of managed D-Bus: Beagle/Xesam

Beagle is working towards supporting the Xesam search specification using managed D-Bus, enabling interoperability between different desktop search systems and user interfaces.
Complex users of managed D-Bus: Banshee

Aaron Bockover's next-generation Banshee backend exposes its media collection framework to the desktop over D-Bus as a re-usable component.
Complex users of managed D-Bus: Telepathy

Telepathy-based IM clients and connection managers have proved to be the acid test for completeness of the binding

Facebook connection manager using Mono.Facebook
Complex users of managed D-Bus: Banter

- Telepathy-based
- Uses telepathy-sharp
- The first application to make heavy use of managed D-Bus thread safety and concurrency features
Applications and bindings (1)

- **telepathy-sharp**
- **Tapioca** VoIP and IM application development framework (tapioca-sharp)
- **Landell** VoIP and IM client using Gtk#
- **Gnome** NetworkManager binding
- **Gnome** Power Manager binding
- **Banshee** provides and uses a media player API, and uses Gnome Power Manager, Gnome NetworkManager, Helix, notify-sharp, org.gnome.SettingsDaemon
- **Helix D-Bus** for remote control of the media stream
- **hal-sharp** is provides access to HAL, the Hardware Abstraction Layer
- **NotifySharp** provides a client implementation for Desktop Notifications and works as a libnotify client replacement
- **F-Spot** personal photo management application, for single-instance detection [1]
- **Tomboy** simple note taking application, for remote control and single-instance detection
- **dcsharp** file sharing client using the Direct Connect protocol, for notifications, single instance and remote control
- **LAT** LDAP Administration Tool
- **VMX Manager**, Virtual Machine Manager), GNOME SVN
Applications and bindings (2)

- **NewStuffManager**, a plugin update/download service
- **last-exit**, a music player for Last.fm
- **Muine**, a music player for GNOME
- **The Fuzz**, process security manager with GUI
- **GShare**, file sharing utility
- **Chatter** (Telepathy GnomeUI) VoIP and IM client using Gtk#
- Babuine **TimeTracker**
- **gnome-keyring-sharp** GNOME Keyring implementation, to get the keyring socket address
- Novell **eIDconfig-belgium** configuration toolkit for the Belgian eID middleware
- **PodSleuth** iPod model information discovery/export tool, using hal-sharp
- Novell **Banter** collaboration client (Telepathy)
- **circ** IRC client
- **Beagle** xesam-adapter desktop search API
- **Many more?**
Get managed D-Bus!

http://www.ndesk.org/DBusSharp

IRC: #managed-dbus / GIMPNet

Managed D-Bus
stable, fast, fun to use