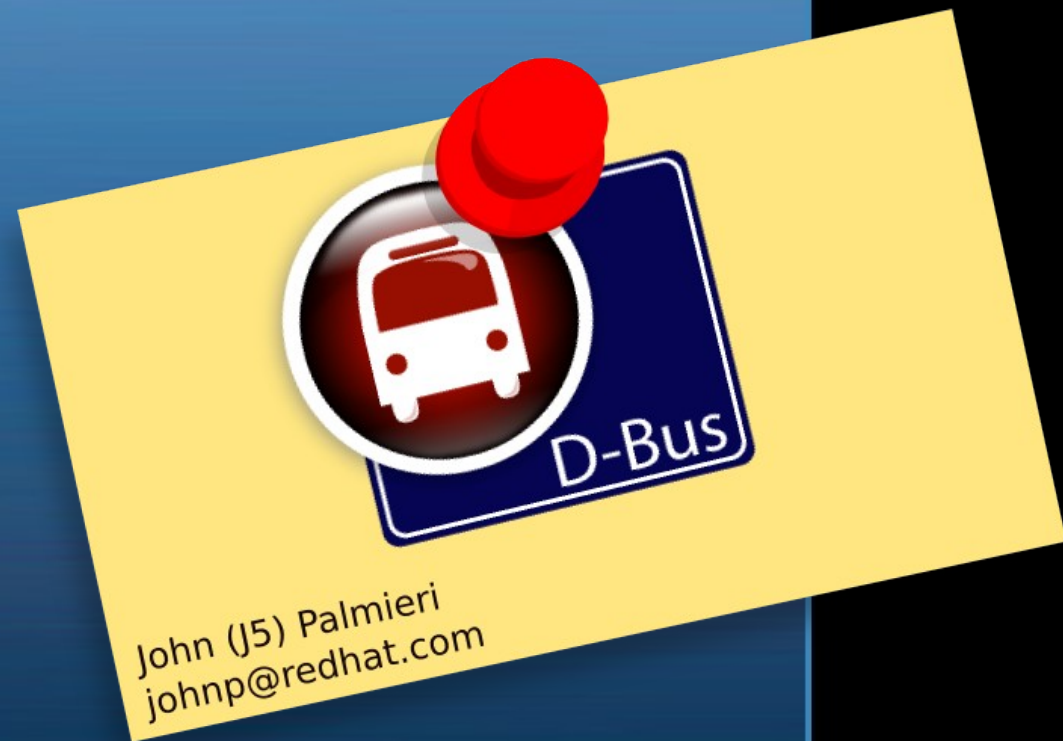


# D-Bus

Debugging and Optimizing your D-Bus Applications



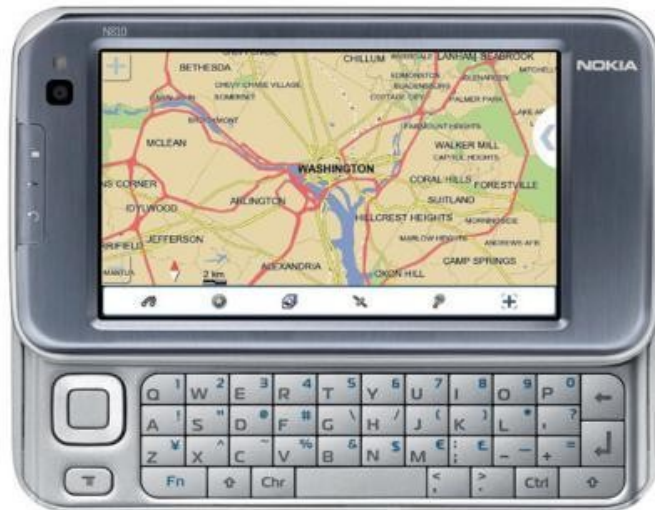
# What is D-Bus?

# D-Bus is...

- A way to talk to other processes
- An interface to find other processes to talk to
- A shared language so processes understand one another



# Some D-Bus consumers



# D-Bus Concepts

# D-Bus Specification

- <http://dbus.freedesktop.org/doc/dbus-specification>
- a protocol
- Multiple transports
- Well defined buses

# Names

- Two types of Names – Bus Name and Unique Name
- A Bus Name is similar to a web address and in fact looks like web addresses in reverse order (e.g. 'org.freedesktop.Hal')
- Unique names are like IP addresses (e.g. ':1.2')
- Names are used to route messages

# Object Path

- Object Paths can be thought of as pointers to objects in the abstract sense
- They look like Unix paths  
(e.g. /org/freedesktop/Hal/Manager)
- Object paths route messages to the correct object within a process which owns a Name

# Interfaces

- Interfaces look just like Bus Names (e.g. org.freedesktop.DBus.Introspectable)
- Interfaces are meant to be used for versioning protocol
- Interfaces route messages to the correct method in an Object Path

# Messages

- D-Bus is a message based IPC
- 4 types of messages can be sent or received
  - `method_call`, `signal`, `method_return` and `error`
- Messages have headers which describe the message
- Messages also contain a body which is a binary stream of parameters

# Message Header

- Contains a bunch of flags and fields describing the message
- Contains a set of optional header fields depending on the message type
  - PATH, INTERFACE, MEMBER, ERROR\_NAME, REPLY\_SERIAL, DESTINATION, SENDER, SIGNATURE

# Message Body

- The message body contains a stream of data defined by the signature field in the header
- These can be thought of as parameters to a method call or return values in the case of method returns

# D-Bus Types

Conventional Name	Code	Description
INVALID	0 (ASCII NUL)	Not a valid type code, used to terminate signatures
BYTE	121 (ASCII 'y')	8-bit unsigned integer
BOOLEAN	98 (ASCII 'b')	invalid.
INT16	110 (ASCII 'n')	16-bit signed integer
UINT16	113 (ASCII 'q')	16-bit unsigned integer
INT32	105 (ASCII 'i')	32-bit signed integer
UINT32	117 (ASCII 'u')	32-bit unsigned integer
INT64	120 (ASCII 'x')	64-bit signed integer
UINT64	116 (ASCII 't')	64-bit unsigned integer
DOUBLE	100 (ASCII 'd')	IEEE 754 double
STRING	115 (ASCII 's')	UTF-8 string (must be valid UTF-8). Must be nul terminated.
OBJECT_PATH	111 (ASCII 'o')	Name of an object instance
SIGNATURE	103 (ASCII 'g')	A type signature
ARRAY	97 (ASCII 'a'), 114 (ASCII 'r'), 40 (ASCII '('),	Array
STRUCT	41 (ASCII ')')	Struct
VARIANT	118 (ASCII 'v')	Variant type (the type of the value is part of the value itself)
DICTIONARY_ENTRY	125 (ASCII '}')	Entry in a dict or map (array of key-value pairs)

# Standard Buses

- There are an unlimited number of buses that can run
- A Bus is defined by its configuration file, lifecycle and how it is discovered
- There are two standard Buses
  - System Bus
  - Session Bus

# Autostarted Names

- Processes can be autostarted based on their Bus Name and a .service file
- If a message is received which is being routed to a Name that does not yet exist it will be held while the Bus searches for Names that match
- There is a patch by David Zuethen to break this out into a daemon so we can autostart system services

# Some D-Bus Issues

# Not Enough Power Captain

- Programmers take the, if it works then it is good approach
- In the first iterations of the 770's software platform processes were waking up every time a signal went across the bus
- The culprit was the NameOwnerChanged signal and promiscuous match rules

# Don't be a Blockhead

- D-Bus bindings make calling methods on the Bus as easy as local calls
- D-Bus is still an IPC layer where method returns can not be guaranteed to return fast
  - Case in point, blocking Introspect calls in the python bindings
- Async calls are harder to deal with but on slower devices the impact can be dramatic

# Try to be Social

- D-Bus defaults to public shared connections to save resources
- Some libraries open up private connections to avoid unintended interactions with the main app
- Opening a connection is the single most wasteful D-Bus operation in terms of number of instructions involved not to mention authentication round trips

# Don't be Ashamed you Ride the Bus

- Some libraries try to hide the fact they use D-Bus
- Poster children – Avahi and HAL
- D-Bus has become a defacto standard, knowing that you are using D-Bus give a developer more control and instant interfaces in many different programming languages

# **D-Bugging the Bus**

# Compiling and running the Bus for Debugging

- `$> ./configure --enable-checks --enable-tests --enable-verbose-mode --enable-asserts`
- `$> DBUS_VERBOSE=1 dbus-daemon --session --nofork --print-address`
- `$> export DBUS_SESSION_BUS_ADDRESS=`
- `$> DBUS_VERBOSE=1 dbus-daemon --system --nofork --print-address`
- `$> export DBUS_SYSTEM_BUS_ADDRESS=`

# Sample Output

```
3980: Checking whether message matches rule
type='4',interface='org.freedesktop.DBus',member='NameO
wnerChanged' for connection 0xb920c5f0
3980: TRANSACTION: executing
3980:   LOCK: dbus_connection_get_data
3980:   UNLOCK: dbus_connection_get_data
3980:   LOCK: dbus_connection_send_preallocated
3980: Message 0xb920d650 (2 no path no interface no
member 'u') for :1.0 added to outgoing queue
0xb920c5f0, 1 pending to send
3980: Message 0xb920d650 serial is 7
3980: _dbus_connection_do_iteration_unlocked start
3980:   UNLOCK: _dbus_connection_acquire_io_path
3980: _dbus_connection_acquire_io_path locking
io_path_mutex
3980: _dbus_connection_acquire_io_path start
connection->io_path_acquired = 0 timeout = 0
3980: _dbus_connection_acquire_io_path end connection-
>io_path_acquired = 1
```

# Use TCP to do remote debugging

- Example:  
`<listen>tcp:host=localhost,port=1234</listen>`
- Back up either `/etc/dbus-1/system.conf` or `/etc/dbus-1/session.conf` and edit the `listen` tag to the above line
- Now you can send messages with `dbus-send`, `D-Feet` or any D-Bus application from a remote machine
- Don't do this in production

# Using dbus-monitor

- dbus-monitor will monitor every message on the bus as long as the configuration specifies the `eavesdrop="true"` on allow policies (default for session bus)
- ```
$> dbus-monitor [--system | --session] [--profile | --monitor] [watch expressions]
```
- Example watch expression:  

```
"type='signal',sender='org.gnome TypingMonitor',interface='org.gnome TypingMonitor'"
```

# dbus-monitor sample output

```
signal sender=org.freedesktop.DBus -> dest=:1.21
path=/org/freedesktop/DBus;
interface=org.freedesktop.DBus; member=NameAcquired
    string ":1.21"
method call sender=:1.21 -> dest=org.freedesktop.DBus
path=/org/freedesktop/DBus;
interface=org.freedesktop.DBus; member=AddMatch
    string "type='method_call'"
method call sender=:1.21 -> dest=org.freedesktop.DBus
path=/org/freedesktop/DBus;
interface=org.freedesktop.DBus; member=AddMatch
    string "type='method_return'"
method call sender=:1.21 -> dest=org.freedesktop.DBus
path=/org/freedesktop/DBus;
interface=org.freedesktop.DBus; member=AddMatch
    string "type='error'"
```

# Using dbus-send

- dbus-send can be used to send messages over the bus
- `dbus-send [--system | --session] [--dest=NAME] [--print-reply] [--type=TYPE] <destination object path> <message name> [contents ...]`
- message name should include the interface
- contents are a list of type/value pairs such as
  - `int32:47 string:'hello world' double:65.32 \`
  - `array:string:"1st item","next item","last item" \`
  - `dict:string:int32:"one",1,"two",2,"three",3 \`
  - `variant:int32:-8 \`
  - `objpath:/org/freedesktop/sample/object/name`

# dbus-send sample output

```
$> dbus-send --session --print-reply \
/org/freedesktop/DBus \
org.freedesktop.DBus.Introspectable.Introspect

method return sender=(null sender) -> dest=(null
destination) reply_serial=2
  string "<!DOCTYPE node PUBLIC "-//freedesktop//DTD
D-BUS Object Introspection 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/introspe
ct.dtd">
<node>
</node>
"
```

# Graphing D-Bus traffic

- <http://alban.apinc.org/blog/index.php/2008/03/08/99-h>

# Using D-Feet to debug



# make check and regression tests

- whenever creating patches for the bus or library itself always run make check to pass the regression test
- whenever adding new functionality you must add a regression test

# Optimization on the Bus

# Digging out blocking calls by simulating slow methods

- This is done easily for D-Bus Python applications by modifying the python module
- Put random sleeps in the exported methods of your compiled applications
- Slowing down methods calls allows you to “see” how potential bottlenecks will effect your program

# Async client side patterns

```
def handle_cb(state_var, param1, param2):  
    <handle the async call like you would signals>  
  
def handle_error_cb(e):  
    <handle this by logging it and/or trying the call  
    again>  
  
state_var = 'I'm a state variable'  
dbus_proxy_obj.some_method_call(param1, param2,  
    reply_handler=lambda param1, param2 :  
        handle_cb(state_var, param1, param2)  
    error_handler=handle_error_cb)
```

# Async isn't just for clients

- services in many bindings have an async reply mode for methods which sends a function into the method to be called once processing is finished
- This allows the service to process other incoming calls while it processes the async call.

# Skip the Bus, go direct

- Applications can become a service and bypass the bus altogether
- This avoids overheads associated with the bus being a middle man
- Drawback is it is more complicated to implement and some bindings do not yet support this
- HAL does it and so does Gnome-VFS

# The Bus is ripe for optimization

- Just look at the TODO's and FIXME's in the code
- Yet to be applied patch makes fixed array validation 10 times faster

# Other Pitfalls

# There is no forking in the Champaign room

- If you fork with a shared connection there is a potential for the socket to be starved of data and messages never reaching the correct process
- Always use private connections and close the socket in the child or ...
- ... unref your shared connections, call `dbus_shutdown` and `dbus_init` again.

# Threads are just as dangerous

- Handle all shared connections in one thread
- Private connections are fine to create in a thread as long as it is not handled outside of that thread

# Listening to other conversations is rude

- Not to mention a waste of resources
- Use D-Bus monitor or log debugging messages every time your app handles a message
- Use the arg and brand new object matching facilities of the bus

# No one cares if you are idle

- A very popular app used to send idle messages once a second. Why? I have no idea.
- Patterns are being discussed for high traffic signals to only emit if a client has registered for it

# Get Your Daily D-Bus Fix

- Web - <http://dbus.freedesktop.org>
- IRC - #dbus on freenode.net
- Mailing list – [dbus@lists.freedesktop.org](mailto:dbus@lists.freedesktop.org)  
<http://lists.freedesktop.org/mailman/listinfo/dbus>