Abteilung für Wirtschaftsinformatik

The ooRexx DBus Bindings for Linux, MacOSX and Windows

The 2016 International Rexx Symposium

Tampa, Florida, USA

Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Welthandelsplatz 1 ■ A-1020 Wien ■ Austria

Agenda

- D-Bus
  - History, usages, concepts, importance has been increasing!
- D-Bus Language Bindings for ooRexx ("dbusoorexx")
  - Overview, marrying a dynamic language with a strictly typed wire protocol
  - Examples for clients, servers (services)
  - Important utility for on-the-fly documentation
  - " dbusdoc.rex" (also "dbusListObjectPaths.rex")
- Roundup and outlook, URLs for further fun

D-Bus History

- History
  - RedHat, Inc.
    - Havoc Pennington
    - First release of the D-Bus specifications: 2003-09-06 (revision 0.8), cf. <http://dbus.freedesktop.org/releases/dbus/>
  - Handed over to "freedesktop.org", cf. <http://dbus.freedesktop.org>
    - Became part of all Linux distributions
    - Latest release version: 1.10.8 (distros often use older versions)
  - Cross-platfrom, ported to other operating systems, e.g.
    - MacOSX
    - Windows

D-Bus
Usages, 1

- Linux kernel communicates with environment
  - Uses the "system" D-Bus daemon (a message broker)
  - Broadcasting D-Bus signals to report noteworthy events
    - E.g. reporting additions/removal of devices
  - For security reasons D-Bus services and interactions are controlled by system service configuration files
  - **Warning:** do *not change the service configuration files* with administrative privileges, if you are not 100% sure what you are doing!
    - You could harm your own system bad time!

D-Bus
Usages, 2

- Applications (services) within sessions
  - Uses the "session" D-Bus daemon (a message broker)
  - Using the user's credentials for using D-Bus services and interactions
  - Allows to interact with D-Bus "session" services using D-Bus messages
  - Allows to control the desktop and many applications
  - Allows to learn about events broadcasted as D-Bus signals from "session" services

D-Bus
Concepts, 1

- D-Bus Transports
  - Unix sockets, address prefix: "unix:"
    - Server and client on same computer
  - launchd, address prefix: "launchd:"
    - Server and client on same computer
  - nonce-TCP/IP sockets, address prefix: "nonce-tcp:"
    - Server and client on same computer
  - TCP/IP sockets, address prefix: "tcp:"
    - Server and client on same *or different* computer

D-Bus
Concepts, 2

- D-Bus Messages
  - Employing a transport, D-Bus messages can be exchanged
  - Message consists of an interface name and a member name
  - There are four message types
    - "call message" that may cause a "reply message" or an "error message" (or no reply at all)
    - a one-way "signal message"
  - Arguments and return values are strictly typed
    - 13 basic types (boolean, byte, double, int16, float, string, ...)
    - 4 container types (array, map/dict, structure, variant)

D-Bus Datatypes

array	a	.Array
boolean	b	Rexx string
byte	y	Rexx string
double	d	Rexx string
int16	n	Rexx string
int32	i	Rexx string
in64	x	Rexx string
objpath	o	Rexx string
signature	g	Rexx string
string	s	Rexx string
uint16	q	Rexx string
uint32	u	Rexx string
uint64	t	Rexx string
unix_fd	h	Rexx string
variant	v	depends on signature
structure	()	.Array
map/dict	a{s…}	.Directory

Some examples:

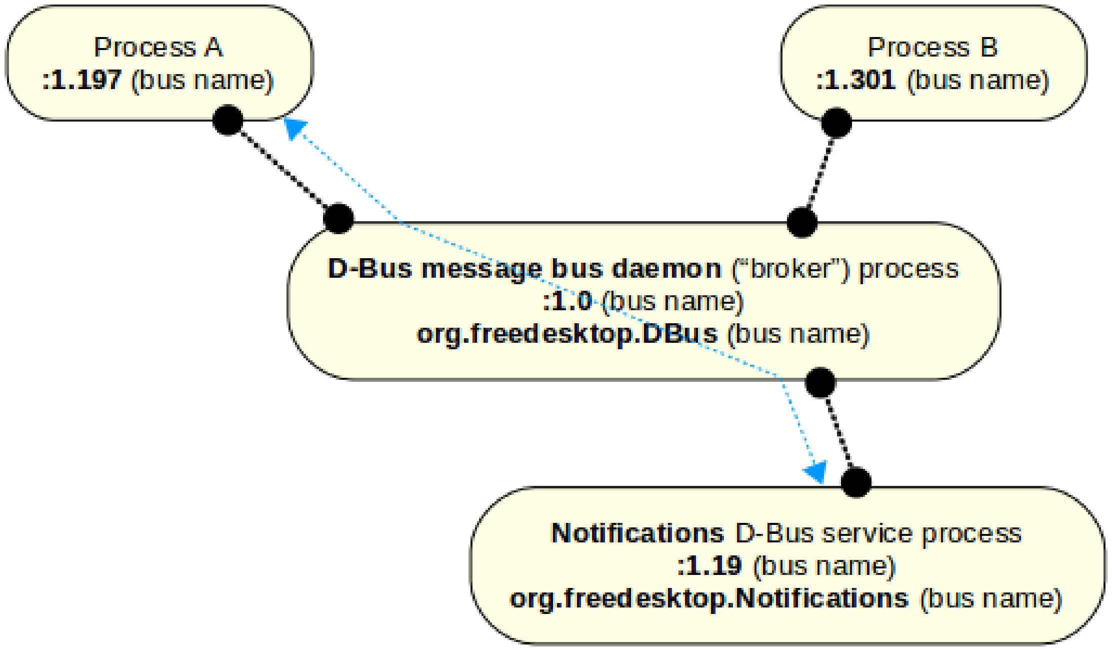
```
org.freedesktop.DBus.Introspectable
s        Introspect()

org.freedesktop.DBus.Properties
v        Get(ss)
a{sv}    GetAll(s)
    Set(ssv)

org.freedesktop.DBus.Notifications
    CloseNotification(u)
as       GetCapabilities()
(ssss)   GetServerInformation()
u        Notify(susssasa{sv}i)
```

…

D-Bus
Concepts, 3

- D-Bus Connection
  - A connection between a D-Bus client and a D-Bus server
  - Dubbed "bus"
- D-Bus Message Daemon/Broker
  - A D-Bus server
  - A set of services that allow it to act as a message broker
    - Relays D-Bus messages among D-Bus clients connected to it
  - Manages D-Bus connections
    - Allows to assign one or more unique names to connections
  - Can start D-Bus services on demand

D-Bus
D-Bus Message Daemon / Broker


Process A
:1.197 (bus name)
Process B
:1.301 (bus name)
D-Bus message bus daemon ("broker") process
:1.0 (bus name)
org.freedesktop.DBus (bus name)
Notifications D-Bus service process
:1.19 (bus name)
org.freedesktop.Notifications (bus name)

D-Bus Concepts, 4

- Object Path
  - A String starting with "/"
  - Denotes the "object" one wishes to send a D-Bus message to
- Sending D-Bus messages
  - Unique bus name, service name
  - Object path
  - Interface name
  - Member name
    - Arguments

D-Bus
Concepts, 5

- Discovering D-Bus service object interfaces on the fly
  - Message org.freedesktop.DBus.Introspectable.Introspect()
    - Returns an XML-encoded file with the interface definitions
  - Addressed to a D-Bus object in a D-Bus service
- Exploited by e.g.
  - dbusoorexx, including dbusdoc.rex
  - d-feet
  - qt-qdbusviewer

D-Bus
Concepts, 6

- Private D-Bus Server
  - Allows to create a simple "private" D-Bus server
    - No daemon/broker services available
  - D-Bus clients can interact with D-Bus server
    - D-Bus infrastructure allows to
      - Connect to a (private) D-Bus server
      - Exchange D-Bus messages with the D-Bus server
  - Makes it easy to create client-server apps fast
    - If using the tcp-transport, then D-Bus based interactions can be across multiple computers!

D-Bus Language Bindings for ooRexx

D-Bus Language Bindings for ooRexx Overview, 1

- Combination of native code ("dbusoorexx", C++) and the ooRexx package named "dbus.cls" (ooRexx)
  - Closely coupled
    - "dbusoorexx" depends on classes and behaviour of "dbus.cls"
    - "dbus.cls", an ooRexx package/program, depends on the features and behaviour of "dbusoorexx", a C++ library

    → *Do not change the code, unless you know what you are doing!*
  - Goals
    - Make it easy for ooRexx programmers to interact with D-Bus
      - Take advantage of a dynamically typed language
      - Apply the Rexx "human-orientation" philosophy where possible

D-Bus Language Bindings for ooRexx Overview, 2

- "dbus.cls"
  - Defines ooRexx classes for the D-Bus language binding
    - DBus
      - Core class to allow
        - Connecting to D-Bus daemons (e.g. "system", "session", address)
        - Sending distinct call and signal messages to D-Bus services
        - Filtering and fetching signal messages from other D-Bus services
        - Getting ooRexx proxy objects for D-Bus service objects
    - DBusProxy
      - Utility class to camouflage a service object as an ooRexx object
        - Returned by .DBus method getObject(busName,objectPath)
      - Automatic method lookup, marshalling of arguments and unmarshalling of return values

D-Bus Language Bindings for ooRexx Overview, 3

- DBusServiceObject
  - Allows ooRexx objects to be used as D-Bus service objects
- DBusSignalListener
  - Implicitly used by .DBus
  - Allows for additional filtering of D-Bus signal messages
- DBusServer
  - Allows to create a private D-Bus server in ooRexx

D-Bus Language Bindings for ooRexx Overview, 4

- IDBusPathMaker
  - Utility class to set up D-Bus service-object discovery for ooRexx DBusServiceObjects
- IntrospectHelper, IntrospectHelperInterface
  - Utility classes to create introspection data on-the-fly
- IDBus, IDBusNode, IDBusInterface, IDBusMethod, IDBusCallMethod, IDBusSignalMethod, IDBusPropertyMethod, IDBusArg, IDBusAnnotation
  - Utility classes for introspection of D-Bus service objects
  - Needed by classes and routines in "dbus.cls"
  - Usually not used by ooRexx programmers

D-Bus Language Bindings for ooRexx Overview, 5

- Public routines
  - dbus.box(signature[,args])
    - Needed for variant values that expect a specific signature
  - stringToUTF8(string)
    - D-Bus string datatype must be UTF-8
    - Converts a Rexx string to UTF-8 (if it contains non-US characters), requires the BSF4ooRexx package
      - Camouflages Java (JRE) as a dynamic, caseless ooRexx class library
      - Cf. <http://bsf4oorexx.sourceforge.net/>
  - DBusDataType(value[,type])
    - Returns the D-Bus datatype name of value, else .nil
    - If type argument given, returns .true or .false, type can be:
      - B[usname], I[nterfaceName] , M[ember], O[bjectPath], S[ignature]

Using D-Bus Services from ooRexx

D-Bus Language Bindings for ooRexx Example 1 (org.freedesktop.Notifications), 1

- Using a common service
  - Bus name ("service name")

    org.freedesktop.Notifications
  - Object path

    /org/freedesktop/Notifications
  - Interface name

    org.freedesktop.Notifications
    - Members

```
        CloseNotification(u)
as      GetCapabilities()
(ssss)  GetServerInformation()
u       Notify(susssasa{sv}i)
```

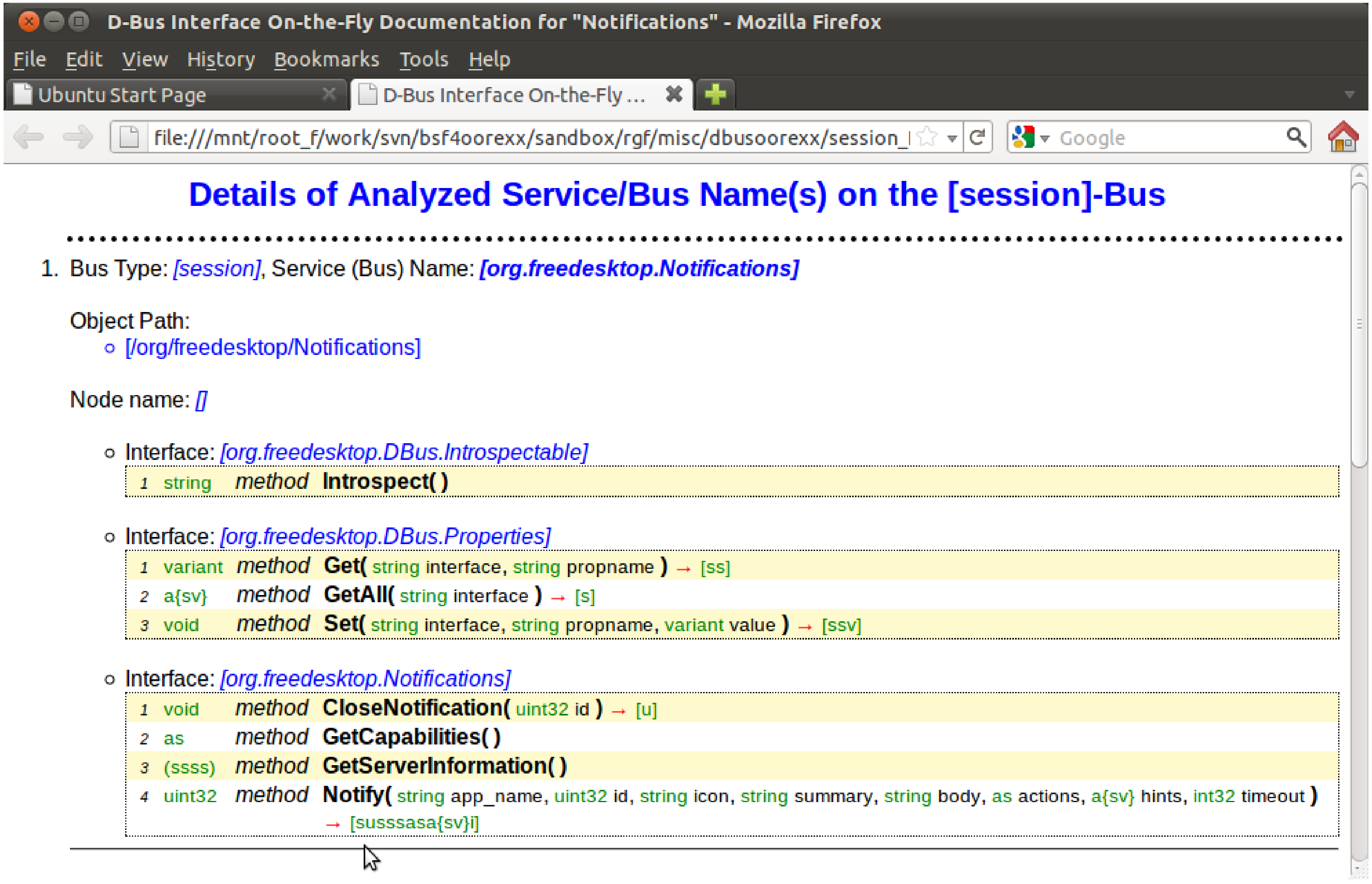
D-Bus Language Bindings for ooRexx Excursion: On-the-fly Documentation, 1

- D-Bus documentation sometimes "meager"
- Idea to exploit the D-Bus infrastructure
  - The "org.freedesktop.DBus" family of interfaces
  - org.freedesktop.DBus.Introspection.Introspect()
    - *Usually* implemented by every D-Bus service objects
- Render interface definitions as HTML text
  - Format results with CSS to allow easy usage, format changes
  - Collect complex signatures and list them at the end
  - Usage:

```
dbusdoc.rex [[session | system] [service name]]
```

D-Bus Language Bindings for ooRexx Excursion: On-the-fly Documentation, 2

```
rexx dbusdoc.rex Notifications
```


D-Bus Interface On-the-Fly Documentation for "Notifications" - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Ubuntu Start Page
D-Bus Interface On-the-Fly ...
file:///mnt/root_f/work/svn/bsf4oorexx/sandbox/rgf/misc/dbusoorexx/session_
Google

Details of Analyzed Service/Bus Name(s) on the [session]-Bus

1. Bus Type: *[session]*, Service (Bus) Name: ***[org.freedesktop.Notifications]***

   Object Path:
   - [/org/freedesktop/Notifications]

   Node name: *[]*

   - Interface: *[org.freedesktop.DBus.Introspectable]*

1	string	*method*	**Introspect( )**

   - Interface: *[org.freedesktop.DBus.Properties]*

1	variant	*method*	**Get(** string interface, string propname **)** → [ss]
2	a{sv}	*method*	**GetAll(** string interface **)** → [s]
3	void	*method*	**Set(** string interface, string propname, variant value **)** → [ssv]

   - Interface: *[org.freedesktop.Notifications]*

1	void	*method*	**CloseNotification(** uint32 id **)** → [u]
2	as	*method*	**GetCapabilities( )**
3	(ssss)	*method*	**GetServerInformation( )**
4	uint32	*method*	**Notify(** string app_name, uint32 id, string icon, string summary, string body, as actions, a{sv} hints, int32 timeout **)** → [susssasa{sv}i]

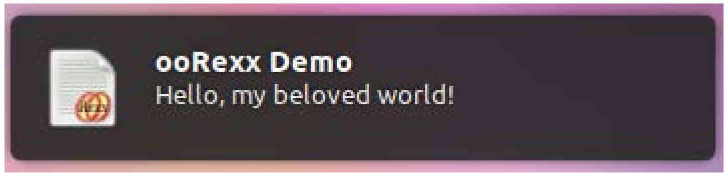
D-Bus Language Bindings for ooRexx
Example 1 (org.freedesktop.Notifications), 2

- Getting the D-Bus service object as an ooRexx object
  - .DBus method getObject(busName,objectPath)
  - returns a DBusProxyObject which
    - Remembers the bus name and the object path
      - Used for sending messages
    - Interrogates the interfaces of the target D-Bus service object
      - Used for automatically determining methods, marshalling arguments and unmarshalling return values

  → Very simple and easy to interact with D-Bus service objects!

D-Bus Language Bindings for ooRexx
Example 1 (org.freedesktop.Notifications), 3

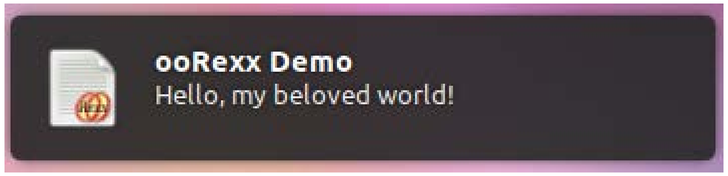
```
 /* get access to remote object */
o=.dbus~session~getObject("org.freedesktop.Notifications","/org/freedesktop/Notifications")

id=o~notify("An ooRexx App", , "oorexx", "ooRexx Demo", "Hello, my beloved world!", , , -1)

.dbus~session~close    /* explicitly close session bus, shuts down DBus message loop thread   */

::requires "dbus.cls"  /* get DBus support  */
```



ooRexx Demo
Hello, my beloved world!

D-Bus Language Bindings for ooRexx Example 1 [Without Proxy], 4

```
conn=.dbus~session      /* get connection to session dbus       */

    /* define message arguments */
busName       ="org.freedesktop.Notifications"
objectName    ="/org/freedesktop/Notifications"
interfaceName ="org.freedesktop.Notifications"
memberName    ="Notify"
replySignature="u"                /* uint32 */
callSignature ="susssasa{sv}i" /* string,uint32,string,string,string,array of string,dict,int32 */

id=conn~message("call",busName,objectName,interfaceName,memberName,replySignature,callSignature, -
                "An ooRexx App", , "oorexx", "ooRexx Demo", "Hello, my beloved world!", , , -1)

conn~close              /* explicitly close connection, shuts down DBus message loop thread */

::requires "dbus.cls"   /* get DBus support  */
```


ooRexx Demo
Hello, my beloved world!

D-Bus Language Bindings for ooRexx
Intercepting DBus Signals

- One-way D-Bus messages with no return value
  - Interface, name, arguments (optional)
- D-Bus daemon allows filtering for them
  - Only forwards matching signal D-Bus messages
- .DBus method listener to add a Rexx listener object
  - Invokes Rexx method named after the signal
  - Otherwise the unknown mechanism of ooRexx can be employed

D-Bus Language Bindings for ooRexx Example 2 (Listening to Signals), 1

```
signal on halt            -- intercept ctl-c (jump to label 'halt:' below)

conn=.dbus~session        -- get the "session" connection
conn~listener("add", .rexxSignalListener~new)   -- add the Rexx listener object
conn~match("add", "type='signal'", .true)       -- ask for any signal message

say "Hit enter to stop listener..."
parse pull answer         -- wait for pressing enter

halt:                     -- a label for a halt condition (ctl-c)
   say "closing connection."
   conn~close             -- close connection, stops message loop

::requires "dbus.cls"        -- get dbus support for ooRexx

::class RexxSignalListener -- just dump all signals/events we receive

 … to be continued on next page …
```

D-Bus Language Bindings for ooRexx Example 2 (Listening to Signals), 2

… continued from previous page …

```
::class RexxSignalListener -- just dump all signals/events we receive
::method unknown           -- this method intercepts all unknown messages
  use arg methName, methArgs

  slotDir=methArgs[methArgs~size]     -- last argument is slotDir
  say "-->" pp(slotDir~messageTypeName) pp(slotDir~interface) -
            pp(slotDir~member)", nrArgs="methArgs~items-1

  if methArgs~items>1 then
  do
     do i=1 to methArgs~items-1
        say "    argument #" i":" pp(methArgs[i])
     end
  end
  say "-"~copies(79)

::method NameOwnerChanged  -- demo how to directly intercept a signal
  use arg name, old, new, slotDir
  say "==> NameOwnerChanged:" "Name:" pp(name)", OldOwner:" pp(old)   -
                              ", NewOwner:" pp(new)
  say "-"~copies(79)

::routine pp      -- "pretty print": enclose string value with square brackets
  parse arg value
  return "["value"]"
```

D-Bus Language Bindings for ooRexx Example 2 (Listening to Signals), 3

```
Running client yields (starting and then clicking into another terminal window):

rony@rony-ThinkPad-X230:~/work/code$ rexx listener.rex
--> [signal] [org.freedesktop.DBus] [NameAcquired], nrArgs=1
    argument # 1: [:1.193]
--------------------------------------------------------------------------------
Hit enter to stop listener...
--> [signal] [org.ayatana.bamf.matcher] [StackingOrderChanged], nrArgs=0
--------------------------------------------------------------------------------
--> [signal] [org.freedesktop.DBus.Properties] [PropertiesChanged], nrArgs=2
    argument # 1: [org.ayatana.bamf.view]
    argument # 2: [a Directory]
--------------------------------------------------------------------------------
--> [signal] [org.ayatana.bamf.view] [ActiveChanged], nrArgs=1
    argument # 1: [1]
--------------------------------------------------------------------------------
--> [signal] [org.ayatana.bamf.matcher] [ActiveWindowChanged], nrArgs=2
    argument # 1: [/org/ayatana/bamf/window/67112584]
    argument # 2: [/org/ayatana/bamf/window/67108875]
--------------------------------------------------------------------------------
--> [signal] [org.freedesktop.DBus.Properties] [PropertiesChanged], nrArgs=2
    argument # 1: [org.ayatana.bamf.view]
    argument # 2: [a Directory]
--------------------------------------------------------------------------------
… cut …
```

Creating D-Bus Services with ooRexx

D-Bus Language Bindings for ooRexx Creating DBus Services ("DBus Server")

- In a nutshell (setting up a D-Bus service)
  - Create a Rexx class that implements the methods
    - Supply a method named Introspect returning XML
  - Request a busname from the D-Bus daemon
  - Define an interface and an object name
- .DBus method serviceObject to add a D-Bus object (object name) and its Rexx service object
  - Invokes Rexx method named after the D-Bus message
  - Otherwise the unknown mechanism of ooRexx can be employed

D-Bus Language Bindings for ooRexx Example 3 (Multiplier, Server/Service), 1

```
#!/usr/bin/rexx
busName       ="org.rexxla.oorexx.demo.Hello"
objectPath    ="/org/rexxla/oorexx/demo/Hello"

signal on halt                 -- intercept ctl-c (jump to label "halt:")
conn=.dbus~session             -- get the session bus
res=conn~busName("request", busName)    -- request a bus name
if res<>.dbus.dir~primaryOwner & res<>.dbus.dir~alreadyOwner then -- o.k., wait for clients?
do
   say "res="res "problem with requesting the bus name" busName", aborting ..."
   exit -1
end

service=.HelloRexxService~new -- create an instance of the Rexx service
conn~serviceObject("add", objectPath, service)  -- add service object to connection

say "Hit enter to stop server..."
parse pull answer

halt:                       -- a ctl-c causes a jump to this label
   say "closing connection."
   conn~close               -- close connection, stops message loop

::requires "dbus.cls"          -- get dbus support for ooRexx

::class HelloRexxService  -- the methods of this class service DBus clients!

 … to be continued on next page …
```

D-Bus Language Bindings for ooRexx Example 3 (Multiplier, Server/Service), 2

```
 … continued from previous page …

::class HelloRexxService  -- the methods of this class service DBus clients!

::method Multiply       -- implementation of "orx.rexxla.oorexx.demo.Hello.Multiply"
   use arg number1, number2, slotDir
   say "Multiply-request received: sender-bus=["slotDir~sender"]" "at=["slotDir~dateTime"]"
   return number1*number2

::method Introspect        /* return the introspection data for this service object */
 return '<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"'  -
        '"http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">'                 -
        '<node>                                                                          '  -
        '  <interface name="org.freedesktop.DBus.Introspectable">                        '  -
        '    <method name="Introspect">                                                  '  -
        '      <arg name="data" direction="out" type="s"/>                              '  -
        '    </method>                                                                   '  -
        '  </interface>                                                                  '  -
        '  <interface name="org.rexxla.oorexx.demo.Hello">                               '  -
        '    <method name="Multiply">                                                    '  -
        '      <arg name="number1" direction="in"  type="d"/>                            '  -
        '      <arg name="number2" direction="in"  type="d"/>                            '  -
        '      <arg name="result"  direction="out" type="d"/>                            '  -
        '    </method>                                                                   '  -
        '  </interface>                                                                  '  -
        '</node>                                                                         '
```

D-Bus Language Bindings for ooRexx Example 3 (Multiplier, Client), 3

```
#!/usr/bin/rexx
conn=.dbus~connect("session") -- connect to the "session" bus
busName       ="org.rexxla.oorexx.demo.Hello"
objectPath    ="/org/rexxla/oorexx/demo/Hello"
o=conn~getObject(busName, objectPath)     -- get the DBus object

parse arg num1 num2                 -- parse command line arguments
if num1="" then num1=12.345         -- supply a default
if num2="" then num2=10.01          -- supply a default
say num1"*"num2"="o~multiply(num1,num2)   -- use the multiply method
conn~close

::requires "dbus.cls"             -- get dbus support for ooRexx
```

```
Running client yields:

rony@rony-ThinkPad-X230:~/work/code$ rexx c_multiplier.rex
12.345*10.01=123.573450
rony@rony-ThinkPad-X230:~/work/code$ rexx c_multiplier.rex 987.65 43.219
987.65*43.219=42685.2453
```

Roundup and Outlook

- Genuine ooRexx language binding for ooRexx
  - 32- and 64-bit ports available
  - Deployable on all Linux systems
- Makes it very easy to exploit D-Bus
  - Rexx philosophy "human-orientness" a guiding principle
  - All D-Bus service objects can be interacted with
  - All D-Bus signals (events) can be handled
- ooRexx D-Bus service objects *easy to implement!*
- Great tools come with it
  - "dbusdoc.rex", "dbusListObjectPaths.rex"
- Support for other D-Bus platforms available!
  - MacOSX
  - Windows

D-Bus Language Bindings for ooRexx URLs as of 2016-08-16

- DBus
  - Defined by *freedesktop.org*
    - <https://www.freedesktop.org/wiki/Software/dbus/>
- Students' works: <http://wi.wu.ac.at/rgf/diplomarbeiten/>
- dbusoorexx (code contains all platforms)
  - Code: <http://sourceforge.net/projects/bsf4oorexx/files/GA/sandbox/dbusoorexx/>
  - Docs: <http://wi.wu.ac.at/rgf/rexx/orx22/201112-DBus4ooRexx-article.pdf>
  - D-Bus for Windows: <http://wi.wu.ac.at/rgf/rexx/orx22/work/>
  - D-Bus for MacOSX, eg. via <http://www.macports.org/>