D-Bus integration in Emacs

This manual documents an API for usage of D-Bus in Emacs. D-Bus is a message bus system, a simple way for applications to talk to one another. An overview of D-Bus can be found at https://dbus.freedesktop.org/.

Next: Inspection, Up: Top [Contents][Index]

1 An overview of D-Bus

D-Bus is an inter-process communication mechanism for applications residing on the same host. The communication is based on *messages*. Data in the messages is carried in a structured way, it is not just a byte stream.

The communication is connection oriented to two kinds of message buses: a so called *system bus*, and a *session bus*. On a given machine, there is always one single system bus for miscellaneous system-wide communication, like changing of hardware configuration. On the other hand, the session bus is always related to a single user’s session.

Every client application, which is connected to a bus, registers under a *unique name* at the bus. This name is used for identifying the client application. Such a unique name starts always with a colon, and looks like ‘`:1.42`’.

Additionally, a client application can register itself to a so called *known name*, which is a series of identifiers separated by dots, as in ‘`org.gnu.Emacs`’. If several applications register to the same known name, these registrations are queued, and only the first application which has registered for the known name is reachable via this name. If this application disconnects from the bus, the next queued unique name becomes the owner of this known name.

An application can install one or several objects under its name. Such objects are identified by an *object path*, which looks similar to paths in a filesystem. An example of such an object path could be ‘`/org/gnu/Emacs/`’.

Applications might send a request to an object, that means sending a message with some data as input parameters, and receiving a message from that object with the result of this message, the output parameters. Such a request is called *method* in D-Bus.

The other form of communication are *signals*. The underlying message is emitted from an object and will be received by all other applications which have registered for such a signal.

All methods and signals an object supports are called *interface* of the object. Interfaces are specified under a hierarchical name in D-Bus; an object can support several interfaces. Such an interface name could be ‘`org.gnu.Emacs.TextEditor`’ or ‘`org.gnu.Emacs.FileManager`’.

Next: Type Conversion, Previous: Overview, Up: Top [Contents][Index]

2 Inspection of D-Bus services.

Next: Bus names, Up: Inspection [Contents][Index]

2.1 D-Bus version.

D-Bus has evolved over the years. New features have been added with new D-Bus versions. There are two variables, which allow the determination of the D-Bus version used.

Variable: dbus-compiled-version

This variable, a string, determines the version of D-Bus Emacs is compiled against. If it cannot be determined the value is `nil`.

Variable: dbus-runtime-version

The other D-Bus version to be checked is the version of D-Bus Emacs runs with. This string can be different from `dbus-compiled-version`. It is also `nil`, if it cannot be determined at runtime.

Next: Introspection, Previous: Version, Up: Inspection [Contents][Index]

2.2 Bus names.

There are several basic functions which inspect the buses for registered names. Internally they use the basic interface '`org.freedesktop.DBus`', which is supported by all objects of a bus.

Function: dbus-list-activatable-names *&optional bus*

This function returns the D-Bus service names, which can be activated for *bus*. It must be either the symbol `:system` (the default) or the symbol `:session`. An activatable service is described in a service registration file. Under GNU/Linux, such files are located at `/usr/share/dbus-1/system-services/` (for the `:system` bus) or `/usr/share/dbus-1/services/`. An activatable service is not necessarily registered at *bus* already.

The result is a list of strings, which is `nil` when there are no activatable service names at all. Example:

```
;; Check, whether the document viewer can be accessed via D-Bus.
(member "org.gnome.evince.Daemon"
        (dbus-list-activatable-names :session))
```

Function: dbus-list-names *bus*

This function returns all service names, which are registered at D-Bus *bus*. The result is a list of strings, which is `nil` when there are no registered service names at all. Well known names are strings like '`org.freedesktop.DBus`'. Names starting with '`:`' are unique names for services.

bus must be either the symbol `:system` or the symbol `:session`.

Function: dbus-list-known-names *bus*

This function retrieves all registered services which correspond to a known name in *bus*. A service has a known name if it doesn't start with '`:`'. The result is a list of strings, which is `nil` when there are no known names at all.

bus must be either the symbol `:system` or the symbol `:session`.

Function: dbus-list-queued-owners *bus service*

For a given service, registered at D-Bus *bus* under the name *service*, this function returns all queued unique names. The result is a list of strings, or `nil` when there are no queued names for *service* at all.

bus must be either the symbol `:system` or the symbol `:session`. *service* must be a known service name as string.

Function: dbus-get-name-owner *bus service*

For a given service, registered at D-Bus *bus* under the name *service*, this function returns the unique name of the name owner. The result is a string, or `nil` when there is no name owner of *service*.

bus must be either the symbol `:system` or the symbol `:session`. *service* must be a known service name as string.

Function: dbus-ping *bus service &optional timeout*

This function checks whether the service name *service* is registered at D-Bus *bus*. If *service* has not yet started, it is autostarted if possible. The result is either `t` or `nil`.

bus must be either the symbol `:system` or the symbol `:session`. *service* must be a string. *timeout*, a nonnegative integer, specifies the maximum number of milliseconds before `dbus-ping` must return. The default value is 25,000. Example:

```
(message
 "%s screensaver on board."
 (cond
  ((dbus-ping :session "org.gnome.ScreenSaver" 100) "Gnome")
  ((dbus-ping :session "org.freedesktop.ScreenSaver" 100) "KDE")
  (t "No")))
```

To check whether *service* is already running without autostarting it, you can instead write:

```
(member service (dbus-list-known-names bus))
```

Function: dbus-get-unique-name *bus*

This function returns the unique name, under which Emacs is registered at D-Bus *bus*, as a string.

bus must be either the symbol `:system` or the symbol `:session`.

Next: Nodes and Interfaces, Previous: Bus names, Up: Inspection [Contents][Index]

2.3 Knowing the details of D-Bus services.

D-Bus services publish their interfaces. This can be retrieved and analyzed during runtime, in order to understand the used implementation.

The resulting introspection data are in XML format. The root introspection element is always a `node` element. It might have a `name` attribute, which denotes the (absolute) object path an interface is introspected.

The root `node` element may have `node` and `interface` children. A child `node` element must have a `name` attribute, this case it is the relative object path to the root `node` element.

An `interface` element has just one attribute, `name`, which is the full name of that interface. The default interface '`org.freedesktop.DBus.Introspectable`' is always present. Example:

```
<node name="/org/bluez">
  <interface name="org.freedesktop.DBus.Introspectable">
    …
  </interface>
  <interface name="org.bluez.Manager">
    …
  </interface>
```

```
  <interface name="org.bluez.Database">
    …
  </interface>
  <interface name="org.bluez.Security">
    …
  </interface>
  <node name="service_audio"/>
  <node name="service_input"/>
  <node name="service_network"/>
  <node name="service_serial"/>
</node>
```

Children of an `interface` element can be `method`, `signal` and `property` elements. A `method` element stands for a D-Bus method of the surrounding interface. The element itself has a `name` attribute, showing the method name. Children elements `arg` stand for the arguments of a method. Example:

```
<method name="ResolveHostName">
  <arg name="interface" type="i" direction="in"/>
  <arg name="protocol" type="i" direction="in"/>
  <arg name="name" type="s" direction="in"/>
  <arg name="aprotocol" type="i" direction="in"/>
  <arg name="flags" type="u" direction="in"/>
  <arg name="interface" type="i" direction="out"/>
  <arg name="protocol" type="i" direction="out"/>
  <arg name="name" type="s" direction="out"/>
  <arg name="aprotocol" type="i" direction="out"/>
  <arg name="address" type="s" direction="out"/>
  <arg name="flags" type="u" direction="out"/>
</method>
```

`arg` elements can have the attributes `name`, `type` and `direction`. The `name` attribute is optional. The `type` attribute stands for the *signature* of the argument in D-Bus. For a discussion of D-Bus types and their Lisp representation see Type Conversion.[1] The `direction` attribute of an `arg` element can be only ‘`in`’ or ‘`out`’; in case it is omitted, it defaults to ‘`in`’.

A `signal` element of an `interface` has a similar structure. The `direction` attribute of an `arg` child element can be only ‘`out`’ here; which is also the default value. Example:

```
<signal name="StateChanged">
  <arg name="state" type="i"/>
  <arg name="error" type="s"/>
</signal>
```

A `property` element has no `arg` child element. It just has the attributes `name`, `type` and `access`, which are all mandatory. The `access` attribute allows the values ‘`readwrite`’, ‘`read`’, and ‘`write`’. Example:

```
<property name="Status" type="u" direction="read"/>
```

`annotation` elements can be children of `interface`, `method`, `signal`, and `property` elements. Unlike properties, which can change their values during lifetime of a D-Bus object, annotations are static. Often they are used for code generators of D-Bus language bindings. Example:

```
<annotation name="de.berlios.Pinot.GetStatistics" value="pinotDBus"/>
```

Annotations have just `name` and `value` attributes, both must be strings.

Function: dbus-introspect *bus service path*

This function returns all interfaces and sub-nodes of *service*, registered at object path *path* at bus *bus*.

bus must be either the symbol `:system` or the symbol `:session`. *service* must be a known service name, and *path* must be a valid object path. The last two parameters are strings. The result, the introspection data, is a string in XML format. Example:

```
(dbus-introspect
 :system "org.freedesktop.Hal"
 "/org/freedesktop/Hal/devices/computer")

⇒ "<!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.Hal.Device">
        <method name="GetAllProperties">
          <arg name="properties" direction="out" type="a{sv}"/>
        </method>
        …
        <signal name="PropertyModified">
          <arg name="num_updates" type="i"/>
          <arg name="updates" type="a(sbb)"/>
        </signal>
      </interface>
      …
    </node>"
```

This example informs us, that the service ‘`org.freedesktop.Hal`’ at object path ‘`/org/freedesktop/Hal/devices/computer`’ offers the interface ‘`org.freedesktop.Hal.Device`’ (and 2 other interfaces not documented here). This interface contains the method ‘`GetAllProperties`’, which needs no input parameters, but returns as output parameter an array of dictionary entries (key-value pairs). Every dictionary entry has a string as key, and a variant as value.

The interface offers also a signal, which returns 2 parameters: an integer, and an array consisting of elements which are a struct of a string and 2 boolean values.[2]

Function: dbus-introspect-xml *bus service path*

This function serves a similar purpose to the function `dbus-introspect`. The returned value is a parsed XML tree, which can be used for further analysis. Example:

```
(dbus-introspect-xml
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main")

⇒ (node ((name . "/org/freedesktop/xesam/searcher/main"))
    (interface ((name . "org.freedesktop.xesam.Search"))
      (method ((name . "GetHitData"))
        (arg ((name . "search")
              (type . "s")
              (direction . "in")))
```

```
            (arg ((name . "hit_ids")
                  (type . "au")
                  (direction . "in")))
            (arg ((name . "fields")
                  (type . "as")
                  (direction . "in")))
            (arg ((name . "hit_data")
                  (type . "aav")
                  (direction . "out"))))
          …
          (signal ((name . "HitsAdded"))
            (arg ((name . "search") (type . "s")))
            (arg ((name . "count") (type . "u")))))
        …)
```

Function: dbus-introspect-get-attribute *object attribute*

This function returns the *attribute* value of a D-Bus introspection *object*. The value of *object* can be any subtree of a parsed XML tree as retrieved with `dbus-introspect-xml`. *attribute* must be a string according to the attribute names in the D-Bus specification. Example:

```
(dbus-introspect-get-attribute
 (dbus-introspect-xml
  :system "org.freedesktop.SystemToolsBackends"
  "/org/freedesktop/SystemToolsBackends/UsersConfig")
 "name")

⇒ "/org/freedesktop/SystemToolsBackends/UsersConfig"
```

If *object* has no *attribute*, the function returns `nil`.

Next: Methods and Signal, Previous: Introspection, Up: Inspection [Contents][Index]

2.4 Detecting object paths and interfaces.

The first elements, to be introspected for a D-Bus object, are further object paths and interfaces.

Function: dbus-introspect-get-node-names *bus service path*

This function returns all node names of *service* in D-Bus *bus* at object path *path* as a list of strings. Example:

```
(dbus-introspect-get-node-names
 :session "org.gnome.seahorse" "/org/gnome/seahorse")

⇒ ("crypto" "keys")
```

The node names stand for further object paths of the D-Bus *service*, relative to *path*. In the example, '`/org/gnome/seahorse/crypto`' and '`/org/gnome/seahorse/keys`' are also object paths of the D-Bus service '`org.gnome.seahorse`'.

Function: dbus-introspect-get-all-nodes *bus service path*

This function returns all node names of *service* in D-Bus *bus* at object path *path*. It returns a list of strings with all object paths of *service*, starting at *path*. Example:

```
(dbus-introspect-get-all-nodes :session "org.gnome.seahorse" "/")

⇒ ("/" "/org" "/org/gnome" "/org/gnome/seahorse"
    "/org/gnome/seahorse/crypto"
    "/org/gnome/seahorse/keys"
    "/org/gnome/seahorse/keys/openpgp"
    "/org/gnome/seahorse/keys/openpgp/local"
    "/org/gnome/seahorse/keys/openssh"
    "/org/gnome/seahorse/keys/openssh/local")
```

Function: dbus-introspect-get-interface-names *bus service path*

This function returns a list strings of all interface names of *service* in D-Bus *bus* at object path *path*. This list will contain the default interface '`org.freedesktop.DBus.Introspectable`'.

Another default interface is '`org.freedesktop.DBus.Properties`'. If present, `interface` elements can also have `property` children. Example:

```
(dbus-introspect-get-interface-names
 :system "org.freedesktop.Hal"
 "/org/freedesktop/Hal/devices/computer")

⇒ ("org.freedesktop.DBus.Introspectable"
    "org.freedesktop.Hal.Device"
    "org.freedesktop.Hal.Device.SystemPowerManagement"
    "org.freedesktop.Hal.Device.CPUFreq")
```

Function: dbus-introspect-get-interface *bus service path interface*

This function returns *interface* of *service* in D-Bus *bus* at object path *path*. The return value is an XML element. *interface* must be a string and a member of the list returned by `dbus-introspect-get-interface-names`. Example:

```
(dbus-introspect-get-interface
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search")

⇒ (interface ((name . "org.freedesktop.xesam.Search"))
    (method ((name . "GetHitData"))
      (arg ((name . "search") (type . "s") (direction . "in")))
      (arg ((name . "hit_ids") (type . "au") (direction . "in")))
      (arg ((name . "fields") (type . "as") (direction . "in")))
      (arg ((name . "hit_data") (type . "aav") (direction . "out"))))
    …
    (signal ((name . "HitsAdded"))
      (arg ((name . "search") (type . "s")))
      (arg ((name . "count") (type . "u")))))
```

With these functions, it is possible to retrieve all introspection data from a running system:

```
(progn
  (pop-to-buffer "*introspect*")
  (erase-buffer)
  (dolist (service (dbus-list-known-names :session))
    (dolist (path (dbus-introspect-get-all-nodes :session service "/"))
      ;; We want to introspect only elements, which have more than
      ;; the default interface "org.freedesktop.DBus.Introspectable".
      (when (delete
```

```
            "org.freedesktop.DBus.Introspectable"
            (dbus-introspect-get-interface-names :session service path))
        (insert (format "\nservice: \"%s\" path: \"%s\"\n" service path)
                (dbus-introspect :session service path))
        (redisplay t)))))
```

Next: Properties and Annotations, Previous: Nodes and Interfaces, Up: Inspection [Contents][Index]

2.5 Applying the functionality.

Methods and signals are the communication means to D-Bus. The following functions return their specifications.

Function: dbus-introspect-get-method-names *bus service path interface*

This function returns a list of strings of all method names of *interface* of *service* in D-Bus *bus* at object path *path*. Example:

```
(dbus-introspect-get-method-names
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search")

⇒ ("GetState" "StartSearch" "GetHitCount" "GetHits" "NewSession"
   "CloseSession" "GetHitData" "SetProperty" "NewSearch"
   "GetProperty" "CloseSearch")
```

Function: dbus-introspect-get-method *bus service path interface method*

This function returns *method* of *interface* as an XML element. It must be located at *service* in D-Bus *bus* at object path *path*. *method* must be a string and a member of the list returned by `dbus-introspect-get-method-names`. Example:

```
(dbus-introspect-get-method
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search" "GetHitData")

⇒ (method ((name . "GetHitData"))
     (arg ((name . "search") (type . "s") (direction . "in")))
     (arg ((name . "hit_ids") (type . "au") (direction . "in")))
     (arg ((name . "fields") (type . "as") (direction . "in")))
     (arg ((name . "hit_data") (type . "aav") (direction . "out"))))
```

Function: dbus-introspect-get-signal-names *bus service path interface*

This function returns a list of strings of all signal names of *interface* of *service* in D-Bus *bus* at object path *path*. Example:

```
(dbus-introspect-get-signal-names
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search")

⇒ ("StateChanged" "SearchDone" "HitsModified"
   "HitsRemoved" "HitsAdded")
```

Function: dbus-introspect-get-signal *bus service path interface signal*

This function returns *signal* of *interface* as an XML element. It must be located at *service* in D-Bus *bus* at object path *path*. *signal* must be a string and a member of the list returned by `dbus-introspect-get-signal-names`. Example:

```
(dbus-introspect-get-signal
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search" "HitsAdded")

⇒ (signal ((name . "HitsAdded"))
     (arg ((name . "search") (type . "s")))
     (arg ((name . "count") (type . "u"))))
```

Next: Arguments and Signatures, Previous: Methods and Signal, Up: Inspection [Contents][Index]

2.6 What else to know about interfaces.

Interfaces can have properties. These can be exposed via the ‘`org.freedesktop.DBus.Properties`’ interface[3]. That is, properties can be retrieved and changed during the lifetime of an element.

A generalized interface is ‘`org.freedesktop.DBus.Objectmanager`’[4], which returns objects, their interfaces and properties for a given service in just one call.

Annotations, on the other hand, are static values for an element. Often, they are used to instruct generators, how to generate code from the interface for a given language binding.

Function: dbus-introspect-get-property-names *bus service path interface*

This function returns a list of strings with all property names of *interface* of *service* in D-Bus *bus* at object path *path*. Example:

```
(dbus-introspect-get-property-names
 :session "org.kde.kded" "/modules/networkstatus"
 "org.kde.Solid.Networking.Client")

⇒ ("Status")
```

If an interface declares properties, the corresponding element supports also the ‘`org.freedesktop.DBus.Properties`’ interface.

Function: dbus-introspect-get-property *bus service path interface property*

This function returns *property* of *interface* as an XML element. It must be located at *service* in D-Bus *bus* at object path *path*. *property* must be a string and a member of the list returned by `dbus-introspect-get-property-names`.

A *property* value can be retrieved by the function `dbus-introspect-get-attribute`. Example:

```
(dbus-introspect-get-property
 :session "org.kde.kded" "/modules/networkstatus"
 "org.kde.Solid.Networking.Client" "Status")

⇒ (property ((access . "read") (type . "u") (name . "Status")))
```

```
(dbus-introspect-get-attribute
 (dbus-introspect-get-property
  :session "org.kde.kded" "/modules/networkstatus"
  "org.kde.Solid.Networking.Client" "Status")
 "access")

⇒ "read"
```

Function: dbus-get-property *bus service path interface property*

This function returns the value of *property* of *interface*. It will be checked at *bus*, *service*, *path*. The result can be any valid D-Bus value, or `nil` if there is no *property*. Example:

```
(dbus-get-property
 :session "org.kde.kded" "/modules/networkstatus"
 "org.kde.Solid.Networking.Client" "Status")

⇒ 4
```

Function: dbus-set-property *bus service path interface property value*

This function sets the value of *property* of *interface* to *value*. It will be checked at *bus*, *service*, *path*. When the value is successfully set, this function returns *value*. Otherwise, it returns `nil`. Example:

```
(dbus-set-property
 :session "org.kde.kaccess" "/MainApplication"
 "com.trolltech.Qt.QApplication" "doubleClickInterval" 500)

⇒ 500
```

Function: dbus-get-all-properties *bus service path interface*

This function returns all properties of *interface*. It will be checked at *bus*, *service*, *path*. The result is a list of cons. Every cons contains the name of the property, and its value. If there are no properties, `nil` is returned. Example:

```
(dbus-get-all-properties
 :session "org.kde.kaccess" "/MainApplication"
 "com.trolltech.Qt.QApplication")

⇒ (("cursorFlashTime" . 1000) ("doubleClickInterval" . 500)
    ("keyboardInputInterval" . 400) ("wheelScrollLines" . 3)
    ("globalStrut" 0 0) ("startDragTime" . 500)
    ("startDragDistance" . 4) ("quitOnLastWindowClosed" . t)
    ("styleSheet" . ""))
```

Function: dbus-get-all-managed-objects *bus service path*

This function returns all objects at *bus*, *service*, *path*, and the children of *path*. The result is a list of objects. Every object is a cons of an existing path name, and the list of available interface objects. An interface object is another cons, whose car is the interface name and cdr is the list of properties as returned by `dbus-get-all-properties` for that path and interface. Example:

```
(dbus-get-all-managed-objects
 :session "org.gnome.SettingsDaemon" "/")

⇒ (("/org/gnome/SettingsDaemon/MediaKeys"
    ("org.gnome.SettingsDaemon.MediaKeys")
```

```
        ("org.freedesktop.DBus.Peer")
        ("org.freedesktop.DBus.Introspectable")
        ("org.freedesktop.DBus.Properties")
        ("org.freedesktop.DBus.ObjectManager"))
       ("/org/gnome/SettingsDaemon/Power"
        ("org.gnome.SettingsDaemon.Power.Keyboard")
        ("org.gnome.SettingsDaemon.Power.Screen")
        ("org.gnome.SettingsDaemon.Power"
         ("Icon" . ". GThemedIcon battery-full-charged-symbolic ")
         ("Tooltip" . "Laptop battery is charged"))
        ("org.freedesktop.DBus.Peer")
        ("org.freedesktop.DBus.Introspectable")
        ("org.freedesktop.DBus.Properties")
        ("org.freedesktop.DBus.ObjectManager"))
       …)
```

If possible, ‘`org.freedesktop.DBus.ObjectManager.GetManagedObjects`’ is used for retrieving the information. Otherwise, the information is collected via ‘`org.freedesktop.DBus.Introspectable.Introspect`’ and ‘`org.freedesktop.DBus.Properties.GetAll`’, which is slow.

An overview of all existing object paths, their interfaces and properties could be retrieved by the following code:

```
(let ((result (mapcar (lambda (service)
                        (cons service
                              (dbus-get-all-managed-objects
                               :session service "/")))
                      (dbus-list-known-names :session))))
  (pop-to-buffer "*objectmanager*")
  (erase-buffer)
  (pp result (current-buffer)))
```

Function: dbus-introspect-get-annotation-names ***bus service path interface &optional name***

This function returns a list of all annotation names as list of strings. If *name* is `nil`, the annotations are children of *interface*, otherwise *name* must be a `method`, `signal`, or `property` XML element, where the annotations belong to. Example:

```
(dbus-introspect-get-annotation-names
 :session "de.berlios.Pinot" "/de/berlios/Pinot"
 "de.berlios.Pinot" "GetStatistics")

⇒ ("de.berlios.Pinot.GetStatistics")
```

Default annotation names[5] are

‘`org.freedesktop.DBus.Deprecated`’

Whether or not the entity is deprecated; defaults to `nil`

‘`org.freedesktop.DBus.GLib.CSymbol`’

The C symbol; may be used for `methods` and `interfaces`

‘`org.freedesktop.DBus.Method.NoReply`’

If set, don’t expect a reply to the `method` call; defaults to `nil`

Function: dbus-introspect-get-annotation *bus service path interface name annotation*

This function returns *annotation* as an XML object. If *name* is `nil`, *annotation* is a child of *interface*, otherwise *name* must be the name of a `method`, `signal`, or `property` XML element, where the *annotation* belongs to.

An attribute value can be retrieved by `dbus-introspect-get-attribute`. Example:

```
(dbus-introspect-get-annotation
 :session "de.berlios.Pinot" "/de/berlios/Pinot"
 "de.berlios.Pinot" "GetStatistics"
 "de.berlios.Pinot.GetStatistics")

⇒ (annotation ((name . "de.berlios.Pinot.GetStatistics")
               (value . "pinotDBus")))

(dbus-introspect-get-attribute
 (dbus-introspect-get-annotation
  :session "de.berlios.Pinot" "/de/berlios/Pinot"
  "de.berlios.Pinot" "GetStatistics"
  "de.berlios.Pinot.GetStatistics")
 "value")

⇒ "pinotDBus"
```

Previous: Properties and Annotations, Up: Inspection [Contents][Index]

2.7 The final details.

Methods and signals have arguments. They are described in the `arg` XML elements.

Function: dbus-introspect-get-argument-names *bus service path interface name*

This function returns a list of all argument names as strings. *name* must be a `method` or `signal` XML element. Example:

```
(dbus-introspect-get-argument-names
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search" "GetHitData")

⇒ ("search" "hit_ids" "fields" "hit_data")
```

Argument names are optional; the function can therefore return `nil`, even if the method or signal has arguments.

Function: dbus-introspect-get-argument *bus service path interface name arg*

This function returns the argument *arg* as an XML object. *name* must be a `method` or `signal` XML element. Example:

```
(dbus-introspect-get-argument
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search" "GetHitData" "search")

⇒ (arg ((name . "search") (type . "s") (direction . "in")))
```

Function: dbus-introspect-get-signature ***bus service path interface name &optional direction***

This function returns the signature of a `method` or `signal`, represented by *name*, as a string.

If *name* is a `method`, *direction* can be either '`in`' or '`out`'. If *direction* is `nil`, '`in`' is assumed.

If *name* is a `signal`, and *direction* is non-`nil`, *direction* must be '`out`'. Example:

```
(dbus-introspect-get-signature
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search" "GetHitData" "in")

⇒ "sauas"

(dbus-introspect-get-signature
 :session "org.freedesktop.xesam.searcher"
 "/org/freedesktop/xesam/searcher/main"
 "org.freedesktop.xesam.Search" "HitsAdded")

⇒ "su"
```

Next: Synchronous Methods, Previous: Inspection, Up: Top [Contents][Index]

3 Mapping Lisp types and D-Bus types.

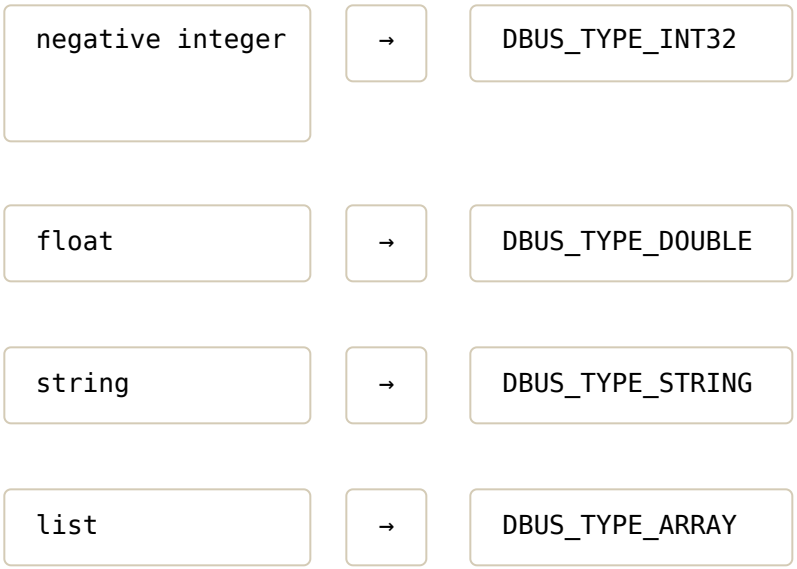
D-Bus method calls and signals accept usually several arguments as parameters, either as input parameter, or as output parameter. Every argument belongs to a D-Bus type.

Such arguments must be mapped between the value encoded as a D-Bus type, and the corresponding type of Lisp objects. The mapping is applied Lisp object → D-Bus type for input parameters, and D-Bus type → Lisp object for output parameters.

3.1 Input parameters.

Input parameters for D-Bus methods and signals occur as arguments of a Lisp function call. The following mapping to D-Bus types is applied, when the corresponding D-Bus message is created:

Lisp type		D-Bus type
t and nil	→	DBUS_TYPE_BOOLEAN
natural number	→	DBUS_TYPE_UINT32



Other Lisp objects, like symbols or hash tables, are not accepted as input parameters.

If it is necessary to use another D-Bus type, a corresponding type symbol can be prepended to the corresponding Lisp object. Basic D-Bus types are represented by the type symbols `:byte`, `:boolean`, `:int16`, `:uint16`, `:int32`, `:uint32`, `:int64`, `:uint64`, `:double`, `:string`, `:object-path`, `:signature` and `:unix-fd`.

Example:

```
(dbus-call-method … nat-number string)
```

is equivalent to

```
(dbus-call-method … :uint32 nat-number :string string)
```

but different to

```
(dbus-call-method … :int32 nat-number :signature string)
```

The value for a byte D-Bus type can be any integer in the range 0 through 255. If a character is used as argument, modifiers represented outside this range are stripped off. For example, `:byte ?x` is equal to `:byte ?\M-x`, but it is not equal to `:byte ?\C-x` or `:byte ?\M-\C-x`. Signed and unsigned integer D-Bus types expect a corresponding integer value.

A D-Bus compound type is always represented as a list. The CAR of this list can be the type symbol `:array`, `:variant`, `:struct` or `:dict-entry`, which would result in a corresponding D-Bus container. `:array` is optional, because this is the default compound D-Bus type for a list.

The objects being elements of the list are checked according to the D-Bus compound type rules.

- An array must contain only elements of the same D-Bus type. It can be empty.
- A variant must contain only a single element.
- A dictionary entry must be an element of an array, and it must contain only a key-value pair of two elements, with a basic D-Bus type key.

- There are no restrictions for structs.

If an empty array needs an element D-Bus type other than string, it can contain exactly one element of D-Bus type `:signature`. The value of this element (a string) is used as the signature of the elements of this array. Example:

```
(dbus-call-method
 :session "org.freedesktop.Notifications"
 "/org/freedesktop/Notifications"
 "org.freedesktop.Notifications" "Notify"
 "GNU Emacs"                   ; Application name.
 0                             ; No replacement of other notifications.
 ""                            ; No icon.
 "Notification summary"        ; Summary.
 (format                       ; Body.
  "This is a test notification, raised from\n%S" (emacs-version))
 '(:array)                     ; No actions (empty array of strings).
 '(:array :signature "{sv}") ; No hints
                               ; (empty array of dictionary entries).
 :int32 -1)                    ; Default timeout.

⇒ 3
```

Function: dbus-string-to-byte-array *string*

Sometimes, D-Bus methods require as input parameter an array of bytes, instead of a string. If it is guaranteed, that *string* is a UTF-8 string, this function performs the conversion. Example:

```
(dbus-string-to-byte-array "/etc/hosts")

⇒ (:array :byte 47 :byte 101 :byte 116 :byte 99 :byte 47
           :byte 104 :byte 111 :byte 115 :byte 116 :byte 115)
```

Function: dbus-escape-as-identifier *string*

This function escapes an arbitrary *string* so it follows the rules for a C identifier. The escaped string can be used as object path component, interface element component, bus name component or member name in D-Bus.

The escaping consists of replacing all non-alphanumerics, and the first character if it’s a digit, with an underscore and two lower-case hex digits. As a special case, "" is escaped to "_". Example:

```
(dbus-escape-as-identifier "0123abc_xyz\x01\xff")

⇒ "_30123abc_5fxyz_01_ff"
```

3.2 Output parameters.

Output parameters of D-Bus methods and signals are mapped to Lisp objects.

D-Bus type	Lisp type

DBUS_TYPE_BOOLEAN	→	t or nil
DBUS_TYPE_BYTE	→	natural number
DBUS_TYPE_UINT16	→	natural number
DBUS_TYPE_INT16	→	integer
DBUS_TYPE_UINT32	→	natural number
DBUS_TYPE_UNIX_FD	→	natural number
DBUS_TYPE_INT32	→	integer
DBUS_TYPE_UINT64	→	natural number
DBUS_TYPE_INT64	→	integer
DBUS_TYPE_DOUBLE	→	float
DBUS_TYPE_STRING	→	string
DBUS_TYPE_OBJECT_PATH	→	string

DBUS_TYPE_SIGNATURE	→	string
DBUS_TYPE_ARRAY	→	list
DBUS_TYPE_VARIANT	→	list
DBUS_TYPE_STRUCT	→	list
DBUS_TYPE_DICT_ENTRY	→	list

The resulting list of the last 4 D-Bus compound types contains as elements the elements of the D-Bus container, mapped according to the same rules.

The signal `PropertyModified`, discussed as an example in Inspection, would offer as Lisp data the following object (*bool* stands here for either `nil` or `t`):

```
(integer ((string bool bool) (string bool bool) …))
```

Function: dbus-byte-array-to-string *byte-array &optional multibyte*

If a D-Bus method or signal returns an array of bytes, which are known to represent a UTF-8 string, this function converts *byte-array* to the corresponding string. The string is unibyte encoded, unless *multibyte* is non-`nil`. Example:

```
(dbus-byte-array-to-string '(47 101 116 99 47 104 111 115 116 115))

⇒ "/etc/hosts"
```

Function: dbus-unescape-from-identifier *string*

This function retrieves the original string from the encoded *string* as a unibyte string. The value of *string* must have been encoded with `dbus-escape-as-identifier`. Example:

```
(dbus-unescape-from-identifier "_30123abc_5fxyz_01_ff")

⇒ "0123abc_xyz\x01\xff"
```

If the original string used in `dbus-escape-as-identifier` is a multibyte string, it cannot be expected that this function returns that string:

```
(string-equal
 (dbus-unescape-from-identifier
  (dbus-escape-as-identifier "Grüß Göttin"))
 "Grüß Göttin")

⇒ nil
```

Next: Asynchronous Methods, Previous: Type Conversion, Up: Top [Contents][Index]

4 Calling methods in a blocking way.

Methods can be called synchronously (*blocking*) or asynchronously (*non-blocking*).

At the D-Bus level, a method call consist of two messages: one message which carries the input parameters to the object owning the method to be called, and a reply message returning the resulting output parameters from the object.

Function: dbus-call-method ***bus service path interface method &optional :timeout timeout &rest args***

This function calls *method* on the D-Bus *bus*. *bus* is either the symbol `:system` or the symbol `:session`.

service is the D-Bus service name to be used. *path* is the D-Bus object path, *service* is registered at. *interface* is an interface offered by *service*. It must provide *method*.

If the parameter `:timeout` is given, the following integer *timeout* specifies the maximum number of milliseconds before the method call must return. The default value is 25,000. If the method call doesn't return in time, a D-Bus error is raised (see Errors and Events).

The remaining arguments *args* are passed to *method* as arguments. They are converted into D-Bus types as described in Type Conversion.

The function returns the resulting values of *method* as a list of Lisp objects, according to the type conversion rules described in Type Conversion. Example:

```
(dbus-call-method
 :session "org.gnome.seahorse" "/org/gnome/seahorse/keys/openpgp"
 "org.gnome.seahorse.Keys" "GetKeyField"
 "openpgp:657984B8C7A966DD" "simple-name")

⇒ (t ("Philip R. Zimmermann"))
```

If the result of the method call is just one value, the converted Lisp object is returned instead of a list containing this single Lisp object. Example:

```
(dbus-call-method
 :system "org.freedesktop.Hal"
 "/org/freedesktop/Hal/devices/computer"
 "org.freedesktop.Hal.Device" "GetPropertyString"
 "system.kernel.machine")

⇒ "i686"
```

With the `dbus-introspect` function it is possible to explore the interfaces of '`org.freedesktop.Hal`' service. It offers the interfaces '`org.freedesktop.Hal.Manager`' for the object at the path

‘/org/freedesktop/Hal/Manager’ as well as the interface ‘org.freedesktop.Hal.Device’ for all objects prefixed with the path ‘/org/freedesktop/Hal/devices’. With the methods ‘GetAllDevices’ and ‘GetAllProperties’, it is simple to emulate the `lshal` command on GNU/Linux systems:

```
(dolist (device
         (dbus-call-method
          :system "org.freedesktop.Hal"
          "/org/freedesktop/Hal/Manager"
          "org.freedesktop.Hal.Manager" "GetAllDevices"))
  (message "\nudi = %s" device)
  (dolist (properties
           (dbus-call-method
            :system "org.freedesktop.Hal" device
            "org.freedesktop.Hal.Device" "GetAllProperties"))
    (message "  %s = %S"
             (car properties) (or (caadr properties) ""))))

-| "udi = /org/freedesktop/Hal/devices/computer
      info.addons = (\"hald-addon-acpi\")
      info.bus = \"unknown\"
      info.product = \"Computer\"
      info.subsystem = \"unknown\"
      info.udi = \"/org/freedesktop/Hal/devices/computer\"
      linux.sysfs_path_device = \"(none)\"
      power_management.acpi.linux.version = \"20051216\"
      power_management.can_suspend_to_disk = t
      power_management.can_suspend_to_ram = \"\"
      power_management.type = \"acpi\"
      smbios.bios.release_date = \"11/07/2001\"
      system.chassis.manufacturer = \"COMPAL\"
      system.chassis.type = \"Notebook\"
      system.firmware.release_date = \"03/19/2005\"
      …"
```

Next: Receiving Method Calls, Previous: Synchronous Methods, Up: Top [Contents][Index]

5 Calling methods non-blocking.

Function: dbus-call-method-asynchronously *bus service path interface method handler &optional :timeout timeout &rest args*

This function calls *method* on the D-Bus *bus* asynchronously. *bus* is either the symbol `:system` or the symbol `:session`.

service is the D-Bus service name to be used. *path* is the D-Bus object path, *service* is registered at. *interface* is an interface offered by *service*. It must provide *method*.

handler is a Lisp function, which is called when the corresponding return message arrives. If *handler* is `nil`, no return message will be expected.

If the parameter `:timeout` is given, the following integer *timeout* specifies the maximum number of milliseconds before a reply message must arrive. The default value is 25,000. If there is no reply message in time, a D-Bus error is raised (see Errors and Events).

The remaining arguments *args* are passed to *method* as arguments. They are converted into D-Bus types as described in Type Conversion.

If *handler* is a Lisp function, the function returns a key into the hash table `dbus-registered-objects-table`. The corresponding entry in the hash table is removed, when the return message arrives, and *handler* is called. Example:

```
(dbus-call-method-asynchronously
 :system "org.freedesktop.Hal"
 "/org/freedesktop/Hal/devices/computer"
 "org.freedesktop.Hal.Device" "GetPropertyString"
 (lambda (msg) (message "%s" msg))
 "system.kernel.machine")

-| i686

⇒ (:serial :system 2)
```

Next: Signals, Previous: Asynchronous Methods, Up: Top [Contents][Index]

6 Offering own methods.

In order to register methods on the D-Bus, Emacs has to request a well known name on the D-Bus under which it will be available for other clients. Names on the D-Bus can be registered and unregistered using the following functions:

Function: dbus-register-service *bus service &rest flags*

This function registers the known name *service* on D-Bus *bus*.

bus is either the symbol `:system` or the symbol `:session`.

service is the service name to be registered on the D-Bus. It must be a known name.

flags is a subset of the following keywords:

`:allow-replacement`

Allow another service to become the primary owner if requested.

`:replace-existing`

Request to replace the current primary owner.

`:do-not-queue`

If we can not become the primary owner do not place us in the queue.

One of the following keywords is returned:

`:primary-owner`

We have become the primary owner of the name *service*.

`:in-queue`

We could not become the primary owner and have been placed in the queue.

`:exists`

We already are in the queue.

`:already-owner`

We already are the primary owner.

Function: dbus-unregister-service *bus service*

This function unregisters all objects from D-Bus *bus*, that were registered by Emacs for *service*.

bus is either the symbol `:system` or the symbol `:session`.

service is the D-Bus service name of the D-Bus. It must be a known name. Emacs releases its association to *service* from D-Bus.

One of the following keywords is returned:

`:released`

We successfully released the name *service*.

`:non-existent`

The name *service* does not exist on the bus.

`:not-owner`

We are not an owner of the name *service*.

When a name has been chosen, Emacs can offer its own methods, which can be called by other applications. These methods could be an implementation of an interface of a well known service, like ‘`org.freedesktop.TextEditor`’.

They could also be an implementation of its own interface. In this case, the service name must be ‘`org.gnu.Emacs`’. The object path shall begin with ‘`/org/gnu/Emacs/`*`application`*’, and the interface name shall be `org.gnu.Emacs.`*`application`*, where *application* is the name of the application which provides the interface.

Constant: dbus-service-emacs

The well known service name ‘`org.gnu.Emacs`’ of Emacs.

Constant: dbus-path-emacs

The object path namespace ‘`/org/gnu/Emacs`’ used by Emacs.

Constant: dbus-interface-emacs

The interface namespace `org.gnu.Emacs` used by Emacs.

Function: dbus-register-method *bus service path interface method handler dont-register-service*

With this function, an application registers *method* on the D-Bus *bus*.

bus is either the symbol `:system` or the symbol `:session`.

service is the D-Bus service name of the D-Bus object *method* is registered for. It must be a known name (see discussion of *dont-register-service* below).

path is the D-Bus object path *service* is registered (see discussion of *dont-register-service* below).

interface is the interface offered by *service*. It must provide *method*.

handler is a Lisp function to be called when a *method* call is received. It must accept as arguments the input arguments of *method*. *handler* should return a list, whose elements are to be used as arguments for the reply message of *method*. This list can be composed like the input parameters in Type Conversion.

If *handler* wants to return just one Lisp object and it is not a cons cell, *handler* can return this object directly, instead of returning a list containing the object.

If *handler* returns a reply message with an empty argument list, *handler* must return the symbol `:ignore`.

When *dont-register-service* is non-`nil`, the known name *service* is not registered. This means that other D-Bus clients have no way of noticing the newly registered method. When interfaces are constructed incrementally by adding single methods or properties at a time, *dont-register-service* can be used to prevent other clients from discovering the still incomplete interface.

The default D-Bus timeout when waiting for a message reply is 25 seconds. This value could be even smaller, depending on the calling client. Therefore, *handler* should not last longer than absolutely necessary.

`dbus-register-method` returns a Lisp object, which can be used as argument in `dbus-unregister-object` for removing the registration for *method*. Example:

```
(defun my-dbus-method-handler (filename)
  (if (find-file filename)
      '(:boolean t)
    '(:boolean nil)))

(dbus-register-method
 :session "org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"
 "org.freedesktop.TextEditor" "OpenFile"
 #'my-dbus-method-handler)

⇒ ((:method :session "org.freedesktop.TextEditor" "OpenFile")
    ("org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"
     my-dbus-method-handler))
```

If you invoke the method ‘`org.freedesktop.TextEditor.OpenFile`’ from another D-Bus application with a file name as parameter, the file is opened in Emacs, and the method returns either *true* or *false*, indicating the success of the method. As a test tool one could use the command line tool `dbus-send` in a shell:

```
# dbus-send --session --print-reply \
    --dest="org.freedesktop.TextEditor" \
    "/org/freedesktop/TextEditor" \
    "org.freedesktop.TextEditor.OpenFile" string:"/etc/hosts"

-| method return sender=:1.22 -> dest=:1.23 reply_serial=2
      boolean true
```

You can indicate an error by raising the Emacs signal `dbus-error`. The handler above could be changed like this:

```
(defun my-dbus-method-handler (&rest args)
  (unless (and (= (length args) 1) (stringp (car args)))
    (signal 'dbus-error (list (format "Wrong argument list: %S" args)))))
```

```
  (condition-case err
      (find-file (car args))
    (error (signal 'dbus-error (cdr err))))
  t)
```

The test then runs

```
# dbus-send --session --print-reply \
    --dest="org.freedesktop.TextEditor" \
    "/org/freedesktop/TextEditor" \
    "org.freedesktop.TextEditor.OpenFile" \
    string:"/etc/hosts" string:"/etc/passwd"

-| Error org.freedesktop.DBus.Error.Failed:
   Wrong argument list: ("/etc/hosts" "/etc/passwd")
```

Function: dbus-register-property ***bus service path interface property access value &optional emits-signal dont-register-service***

With this function, an application declares a *property* on the D-Bus *bus*.

bus is either the symbol `:system` or the symbol `:session`.

service is the D-Bus service name of the D-Bus. It must be a known name.

path is the D-Bus object path *service* is registered (see discussion of *dont-register-service* below).

interface is the name of the interface used at *path*, *property* is the name of the property of *interface*.

access indicates, whether the property can be changed by other services via D-Bus. It must be either the symbol `:read` or `:readwrite`. *value* is the initial value of the property, it can be of any valid type (See dbus-call-method, for details).

If *property* already exists on *path*, it will be overwritten. For properties with access type `:read` this is the only way to change their values. Properties with access type `:readwrite` can be changed by `dbus-set-property`.

The interface '`org.freedesktop.DBus.Properties`' is added to *path*, including a default handler for the '`Get`', '`GetAll`' and '`Set`' methods of this interface. When *emits-signal* is non-`nil`, the signal '`PropertiesChanged`' is sent when the property is changed by `dbus-set-property`.

When *dont-register-service* is non-`nil`, the known name *service* is not registered. This means that other D-Bus clients have no way of noticing the newly registered method. When interfaces are constructed incrementally by adding single methods or properties at a time, *dont-register-service* can be used to prevent other clients from discovering the still incomplete interface.

Example:

```
(dbus-register-property
 :session "org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"
 "org.freedesktop.TextEditor" "name" :read "GNU Emacs")

⇒ ((:property :session "org.freedesktop.TextEditor" "name")
    ("org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"))

(dbus-register-property
 :session "org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"
 "org.freedesktop.TextEditor" "version" :readwrite emacs-version t)
```

```
⇒ ((:property :session "org.freedesktop.TextEditor" "version")
    ("org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"))
```

Other D-Bus applications can read the property via the default methods ‘`org.freedesktop.DBus.Properties.Get`’ and ‘`org.freedesktop.DBus.Properties.GetAll`’. Testing is also possible via the command line tool `dbus-send` in a shell:

```
# dbus-send --session --print-reply \
    --dest="org.freedesktop.TextEditor" \
    "/org/freedesktop/TextEditor" \
    "org.freedesktop.DBus.Properties.GetAll" \
    string:"org.freedesktop.TextEditor"

-| method return sender=:1.22 -> dest=:1.23 reply_serial=3
      array [
         dict entry(
            string "name"
            variant             string "GNU Emacs"
         )
         dict entry(
            string "version"
            variant             string "23.1.50.5"
         )
      ]
```

It is also possible to apply the `dbus-get-property`, `dbus-get-all-properties` and `dbus-set-property` functions (see Properties and Annotations).

```
(dbus-set-property
 :session "org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"
 "org.freedesktop.TextEditor" "version" "23.1.50")

⇒ "23.1.50"

(dbus-get-property
 :session "org.freedesktop.TextEditor" "/org/freedesktop/TextEditor"
 "org.freedesktop.TextEditor" "version")

⇒ "23.1.50"
```

Function: dbus-unregister-object *object*

This function unregisters *object* from the D-Bus. *object* must be the result of a preceding `dbus-register-method`, `dbus-register-property` or `dbus-register-signal` call (see Signals). It returns `t` if *object* has been unregistered, `nil` otherwise.

When *object* identifies the last method or property, which is registered for the respective service, Emacs releases its association to the service from D-Bus.

Next: Alternative Buses, Previous: Receiving Method Calls, Up: Top [Contents][Index]

7 Sending and receiving signals.

Signals are one way messages. They carry input parameters, which are received by all objects which have registered for such a signal.

Function: dbus-send-signal *bus service path interface signal &rest args*

This function is similar to `dbus-call-method`. The difference is, that there are no returning output parameters.

The function emits *signal* on the D-Bus *bus*. *bus* is either the symbol `:system` or the symbol `:session`. It doesn't matter whether another object has registered for *signal*.

Signals can be unicast or broadcast messages. For broadcast messages, *service* must be `nil`. Otherwise, *service* is the D-Bus service name the signal is sent to as a unicast message.[6] *path* is the D-Bus object path *signal* is sent from. *interface* is an interface available at *path*. It must provide *signal*.

The remaining arguments *args* are passed to *signal* as arguments. They are converted into D-Bus types as described in Type Conversion. Example:

```
(dbus-send-signal
 :session nil dbus-path-emacs
 (concat dbus-interface-emacs ".FileManager") "FileModified"
 "/home/albinus/.emacs")
```

Function: dbus-register-signal *bus service path interface signal handler &rest args*

With this function, an application registers for a signal on the D-Bus *bus*.

bus is either the symbol `:system` or the symbol `:session`.

service is the D-Bus service name used by the sending D-Bus object. It can be either a known name or the unique name of the D-Bus object sending the signal. A known name will be mapped onto the unique name of the object, owning *service* at registration time. When the corresponding D-Bus object disappears, signals will no longer be received.

path is the corresponding D-Bus object path that *service* is registered at. *interface* is an interface offered by *service*. It must provide *signal*.

service, *path*, *interface* and *signal* can be `nil`. This is interpreted as a wildcard for the respective argument.

handler is a Lisp function to be called when the *signal* is received. It must accept as arguments the output parameters *signal* is sending.

The remaining arguments *args* can be keywords or keyword string pairs.[7] Their meaning is as follows:

`:arg`*`n`* *`string`*

`:path`*`n`* *`string`*

This stands for the *n*th argument of the signal. `:path`*n* arguments can be used for object path wildcard matches as specified by D-Bus, while an `:argN` argument requires an exact match.

`:arg-namespace` ***`string`***

Register for those signals, whose first argument names a service or interface within the namespace *string*.

`:path-namespace` ***`string`***

Register for the object path namespace *string*. All signals sent from an object path, which has *string* as the preceding string, are matched. This requires *path* to be `nil`.

`:eavesdrop`

Register for unicast signals which are not directed to the D-Bus object Emacs is registered at D-Bus BUS, if the security policy of BUS allows this. Otherwise, this argument is ignored.

`dbus-register-signal` returns a Lisp object, which can be used as argument in `dbus-unregister-object` for removing the registration for *signal*. Example:

```
(defun my-dbus-signal-handler (device)
  (message "Device %s added" device))

(dbus-register-signal
 :system "org.freedesktop.Hal" "/org/freedesktop/Hal/Manager"
 "org.freedesktop.Hal.Manager" "DeviceAdded"
 #'my-dbus-signal-handler)

⇒ ((:signal :system "org.freedesktop.Hal.Manager" "DeviceAdded")
    ("org.freedesktop.Hal" "/org/freedesktop/Hal/Manager"
     my-signal-handler))
```

As we know from the introspection data of interface '`org.freedesktop.Hal.Manager`', the signal '`DeviceAdded`' provides one single parameter, which is mapped into a Lisp string. The callback function `my-dbus-signal-handler` must therefore define a single string argument. Plugging a USB device into your machine, when registered for signal '`DeviceAdded`', will show you which objects the GNU/Linux `hal` daemon adds.

Some of the match rules have been added to a later version of D-Bus. In order to test the availability of such features, you could register for a dummy signal, and check the result:

```
(dbus-ignore-errors
  (dbus-register-signal
   :system nil nil nil nil #'ignore :path-namespace "/invalid/path"))

⇒ nil
```

Next: Errors and Events, Previous: Signals, Up: Top [Contents][Index]

8 Alternative buses and environments.

Until now, we have spoken about the system and the session buses, which are the default buses to be connected to. However, it is possible to connect to any bus with a known address. This is a UNIX domain or TCP/IP socket. Everywhere, where a *bus* is mentioned as argument of a function (the symbol `:system` or the symbol `:session`), this address can be used instead. The connection to this bus must be initialized first.

Function: dbus-init-bus *bus &optional private*

This function establishes the connection to D-Bus *bus*.

bus can be either the symbol `:system` or the symbol `:session`, or it can be a string denoting the address of the corresponding bus. For the system and session buses, this function is called when loading `dbus.el`, there is no need to call it again.

The function returns the number of connections this Emacs session has established to the *bus* under the same unique name (see dbus-get-unique-name). It depends on the libraries Emacs is linked with, and on the environment Emacs is running. For example, if Emacs is linked with the GTK+ toolkit, and it runs in a GTK+-aware environment like GNOME, another connection might already be established.

When *private* is non-`nil`, a new connection is established instead of reusing an existing one. It results in a new unique name at the bus. This can be used, if it is necessary to distinguish from another connection used in the same Emacs process, like the one established by GTK+. It should be used with care for at least the `:system` and `:session` buses, because other Emacs Lisp packages might already use this connection to those buses.

Example: You initialize a connection to the AT-SPI bus on your host:

```
(setq my-bus
      (dbus-call-method
       :session "org.a11y.Bus" "/org/a11y/bus"
       "org.a11y.Bus" "GetAddress"))

⇒ "unix:abstract=/tmp/dbus-2yzWHOCdSD,guid=a490dd26625870ca1298b6e10000fd7f"

;; If Emacs is built with GTK+ support, and you run in a GTK+-enabled
;; environment (like a GNOME session), the initialization reuses the
;; connection established by GTK+'s atk bindings.
(dbus-init-bus my-bus)

⇒ 2

(dbus-get-unique-name my-bus)

⇒ ":1.19"

;; Open a new connection to the same bus.  This supersedes the
;; previous one.
(dbus-init-bus my-bus 'private)

⇒ 1

(dbus-get-unique-name my-bus)

⇒ ":1.20"
```

D-Bus addresses can specify a different transport. A possible address could be based on TCP/IP sockets, see next example. Which transport is supported depends on the bus daemon configuration, however.

Function: dbus-setenv *bus variable value*

This function sets the value of the *bus* environment *variable* to *value*.

bus is either a Lisp symbol, `:system` or `:session`, or a string denoting the bus address. Both *variable* and *value* should be strings.

Normally, services inherit the environment of the bus daemon. This function adds to or modifies that environment when activating services.

Some bus instances, such as `:system`, may disable setting the environment. In such cases, or if this feature is not available in older D-Bus versions, this function signals a `dbus-error`.

As an example, it might be desirable to start X11 enabled services on a remote host's bus on the same X11 server the local Emacs is running. This could be achieved by

```
(setq my-bus "unix:host=example.gnu.org,port=4711")

⇒ "unix:host=example.gnu.org,port=4711"

(dbus-init-bus my-bus)

⇒ 1

(dbus-setenv my-bus "DISPLAY" (getenv "DISPLAY"))

⇒ nil
```

Next: Index, Previous: Alternative Buses, Up: Top [Contents][Index]

9 Errors and events.

The internal actions can be traced by running in a debug mode.

Variable: dbus-debug

If this variable is non-`nil`, D-Bus specific debug messages are raised.

Input parameters of `dbus-call-method`, `dbus-call-method-asynchronously`, `dbus-send-signal`, `dbus-register-method`, `dbus-register-property` and `dbus-register-signal` are checked for correct D-Bus types. If there is a type mismatch, the Lisp error `wrong-type-argument` `D-Bus` *`arg`* is raised.

All errors raised by D-Bus are signaled with the error symbol `dbus-error`. If possible, error messages from D-Bus are appended to the `dbus-error`.

Special Form: dbus-ignore-errors *forms…*

This executes *forms* exactly like a `progn`, except that `dbus-error` errors are ignored during the *forms*. These errors can be made visible when `dbus-debug` is set to `t`.

Incoming D-Bus messages are handled as Emacs events, see (elisp)Misc Events. They are retrieved only, when Emacs runs in interactive mode. The generated event has this form:

```
(dbus-event bus type serial service path interface member handler
        &rest args)
```

bus identifies the D-Bus the message is coming from. It is either the symbol `:system` or the symbol `:session`.

type is the D-Bus message type which has caused the event. It can be `dbus-message-type-invalid`, `dbus-message-type-method-call`, `dbus-message-type-method-return`, `dbus-message-type-error`, or `dbus-message-type-signal`. *serial* is the serial number of the received D-Bus message.

service and *path* are the unique name and the object path of the D-Bus object emitting the message. *interface* and *member* denote the message which has been sent.

handler is the callback function which has been registered for this message (see Signals). When a `dbus-event` event arrives, *handler* is called with *args* as arguments.

In order to inspect the `dbus-event` data, you could extend the definition of the callback function in Signals:

```
(defun my-dbus-signal-handler (&rest args)
  (message "my-dbus-signal-handler: %S" last-input-event))
```

There exist convenience functions which could be called inside a callback function in order to retrieve the information from the event.

Function: dbus-event-bus-name *event*

This function returns the bus name *event* is coming from. The result is either the symbol `:system` or the symbol `:session`.

Function: dbus-event-message-type *event*

This function returns the message type of the corresponding D-Bus message. The result is a natural number.

Function: dbus-event-serial-number *event*

This function returns the serial number of the corresponding D-Bus message. The result is a natural number.

Function: dbus-event-service-name *event*

This function returns the unique name of the D-Bus object *event* is coming from.

Function: dbus-event-path-name *event*

This function returns the object path of the D-Bus object *event* is coming from.

Function: dbus-event-interface-name *event*

This function returns the interface name of the D-Bus object *event* is coming from.

Function: dbus-event-member-name *event*

This function returns the member name of the D-Bus object *event* is coming from. It is either a signal name or a method name.

D-Bus errors are not propagated during event handling, because it is usually not desired. D-Bus errors in events can be made visible by setting the variable `dbus-debug` to `t`. They can also be handled by a hook function.

Variable: dbus-event-error-functions

This hook variable keeps a list of functions, which are called when a D-Bus error happens in the event handler. Every function must accept two arguments, the event and the error variable caught in `condition-case` by `dbus-error`.

Such functions can be used to adapt the error signal to be raised. Example:

```
(defun my-dbus-event-error-handler (event error)
  (when (string-equal (concat dbus-interface-emacs ".FileManager")
```

```
                        (dbus-event-interface-name event))
          (message "my-dbus-event-error-handler: %S %S" event error)
          (signal 'file-error (cdr error))))

      (add-hook 'dbus-event-error-functions #'my-dbus-event-error-handler)
```

Hook functions should take into account that there might be other D-Bus applications running. They should therefore check carefully, whether a given D-Bus error is related to them.

Next: GNU Free Documentation License, Previous: Errors and Events, Up: Top [Contents][Index]

Index

E

I

M

O

R

S

T

Previous: Index, Up: Top [Contents][Index]

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a

licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

3. State on the Title page the name of the publisher of the Modified Version, as the publisher.

4. Preserve all the copyright notices of the Document.

5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

8. Include an unaltered copy of this License.

9. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

11. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

13. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

14. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    “Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

    “CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    “Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with…Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Footnotes

(1)

D-Bus signatures are explained in the D-Bus specification https://dbus.freedesktop.org/doc/dbus-specification.html#message-protocol-signatures.

(2)

The interfaces of the service ‘`org.freedesktop.Hal`’ are described in the HAL specification.

(3)

See https://dbus.freedesktop.org/doc/dbus-specification.html#standard-interfaces-properties

(4)

See https://dbus.freedesktop.org/doc/dbus-specification.html#standard-interfaces-objectmanager

(5)

See https://dbus.freedesktop.org/doc/dbus-specification.html#introspection-format

(6)

For backward compatibility, a broadcast message is also emitted if *service* is the known or unique name Emacs is registered at D-Bus *bus*.

(7)

For backward compatibility, the arguments *args* can also be just strings. They stand for the respective arguments of *signal* in their order, and are used for filtering as well. A `nil` argument might be used to preserve the order.
