The Bonobo Configuration System

Dietmar Maurer

The Bonobo Configuration System

by Dietmar Maurer

Table of Contents

List of Tables

Chapter 1. Bonobo Configuration System Features

Introduction

The Bonobo Configuration System (BCS) consists of several parts. An API to access configuration data, a database to store configuration values in XML format and a system to visualise and edit configuration data. The whole system is built on top of bonobo and ORBit (CORBA).

There are several APIs to access the configuration data, and the API can be chosen through the bonobo moniker system. It is up to the programmer to decide which interface is best for a given application.

The configuration system allows you to store the data with various backends. Although BCS is shipped with it's own XML based backend, it is also possible to use GConf, or LDAP as backend.

Some configuration systems only permit you to store a limited set of types. We have removed that limitation so that we can now store CORBA:any which is very convenient in some situations.

Configuration Databases

Configuration databases are the low level API to access configuration data. All other APIs are build on top of this one. There are some handy C wrapper functions in case you don't want to use the CORBA interface. This also provides a smooth migration path if you previously used GConf or the old gnome-config system.

As mentioned above BCS allows you to use various kinds of databases to actually store the data. A convenient way is to use the XML based standard database, called "xmldb". This format is human readable and also provides a way to store localised values. It can be used together with the xml-i18n-tools.

There is a simple plug-in mechanism for database backends - namely monikers. For example you can access "xmldb" databases through the "xmldb:" moniker, and the GConf database is available through the "gconf:" moniker.

Directory Hierarchy and Names

BCS can store a whole configuration tree, just like a file system. Each directory can contain zero or more

entries (configuration values). Directories and entries use a different name space, so there are no conflicts if a directory and a value use the same name.

We use a slash (`/`) as directory delimiter, and names may not contain colons (`:`).

Default Values

Most applications wants to provide some default values for configuration data, which can then be overwritten by the user. Default values are most often stored in some system directory (read only), whereas the user specific data is stored in the users home directory. For example an application stores default values in `/usr/etc/appname.xmldb`, and user specified value in `~/.appname.xmldb`.

Another scenario is that the default values are stored somewhere at a LDAP server, or there is a group-ware server which provides default values. In both cases user specified values can either be stored in the users home directory, or at the LDAP/group-ware server.

To reach a maximum of flexibility BCS allows you to combine databases. That way is possible to add one database to another, which is then used to lookup default values.

Documentation for values

BCS does not treat documentation as special data type. Documentation is handled like any other (string) data. The convention is to store all documentations inside the `/doc/` directory, so that one can lookup the documentation for `$(key)` at `/doc/$(key)`.

Notifications

BCS also supports a data change notification service. All interested applications are notified if configuration data is changed. This makes it possible that configuration takes effect on-the-fly without restarting any application.

We use the BonoboEventSource interface to implement notifications. Each ConfigDatabase is aggregated with such an event source, and emits the following events:

`Bonobo/Property:change:$(key)`

You should listen to those events if you are interested in changes somewhere in the hierarchy below $(key)

`Bonobo/ConfigDatabase:change$(dir):$(name)`

This format is well suited if you want to listen to changes in a specified directory $(dir), or listen to changes of a specific value.

A common situation is where you want to listen to all changes inside a single directory. You can use the following mask to listen to changes in directory `/testdir`:

```
mask = "Bonobo/ConfigDatabase:change/testdir:";

bonobo_event_source_client_add_listener (db, change_cb, mask, ev, NULL);
```

If you want to listen to a specific value, say "value1", you can use:

```
mask = "Bonobo/ConfigDatabase:change/testdir:value1";
```

Monikers

You can access the whole configuration system with monikers. Monikers are used to name objects, and they effectively implement an object naming space. The following table contains the moniker names together with the interfaces they are able to resolve. Some of the monikers are also able to use parent monikers if present.

Table 1-1. BCS Monikers

moniker name	parent moniker interface	provided interfaces
xmldb:	Bonobo/ConfigDatabase	Bonobo/ConfigDatabase
gconf:	-	Bonobo/ConfigDatabase
config: Bonobo/PropertyBag Bonobo/Control Bonobo/PropertyEditor Bonobo/ConfigDatabase	Bonobo/ConfigDatabase	Bonobo/Property

You can get configuration database interfaces with the "xmldb:" and "gconf:" monikers. Additional interfaces are provided by the "config:" moniker. Here are some examples on how to use those monikers.

If you want direct access to a xml file containing configuration data you can use:

```
Bonobo_ConfigDatabase db;
```

```
db =  bonobo_get_object ("xmldb:~/test.xmldb",
                 Bonobo/ConfigDatabase", ev);
```

You can then pass that database object to other functions to get and set values:

```
autosave = bonobo_config_get_bool (db, "/gnumeric/autosave", ev);

bonobo_config_set_bool (db, "/gnumeric/autosave", FALSE, ev);
```

As mentioned above you can use a parent moniker with the xmldb moniker. The parent is then used to lookup default values:

```
Bonobo_ConfigDatabase db;
char *moniker_name;

moniker_name = "xmldb:~/test-defaults.xmldb#xmldb:~/test.xmldb";
db =  bonobo_get_object (moniker_name, Bonobo/ConfigDatabase", ev);
```

The configuration moniker is invoked by using the "config:" prefix. The string afterwards is the configuration locator. The moniker support being queried against the "Bonobo/Property" or "Bonobo/PropertyBag" depending on whether we are requesting a set of attributes, or a single attribute. You can also query for a "Bonobo/Control" if you want to display or edit the value within your application.

For example, retrieving the configuration information for a specific configuration property in Gnumeric would work like this:

```
Bonobo_Property auto_save;
CORBA_Any value;

auto_save = bonobo_get_object ("config:gnumeric/auto-save",
                       "IDL:Bonobo/Property:1.0", ev);
value = bonobo_property_get_value (auto_save, ev);
```

Bonobo Controls and Property Editors

As already mentioned the "config:" moniker can be queried for a "Bonobo/Control" or "Bonobo/PropertyEditor". BCS uses a slightly extended version of Controls, called PropertyEditors. A PropertyEditor is simply a Control which is able to edit and view properties.

BCS comes with a set of PropertyEditors to edit all basic types, and there are default editors for structures and lists. This way you can display and edit values with arbitrary types. It is also possible to develop your own PropertyEditor.

Here is the code to get a widget which is able to edit a configuration value:

```
Bonobo_Control  control;
GtkWidget      *widget;

control = bonobo_get_object ("config:gnumeric/auto-save",
                     "IDL:Bonobo/Control:1.0", ev);

widget = bonobo_widget_new_control_from_objref (control, NULL);
```

Another common scenario is that you need a property editor for a property contained in a PropertyBag. The following code returns a widget which is able to edit the property *"name"* within the PropertyBag *bag*. The type of the property has to be *TC_long*.

```
widget = bonobo_config_widget_new (TC_long, bag, "name", NULL, ev);
```

Preferences Dialogs

Almost every application needs a prferences dialog. BCS supports the automatic creation of such dialogs from PropertyControls:

```
Bonobo_ConfigControl config_control;
GtkWidget            *dialog;

dialog = bonobo_preferences_new (config_control);
gtk_widget_show (dialog);
```

So the real question is how to create such PropertyControls. On way is to use the standard Bonobo PropertyControl implementation. Another, and by far simpler way is to use a BonoboConfigControl object:

```
BonoboConfigControl *config_control;
Bonobo_PropertyBag   pb;
GtkWidget           *dialog;

config_control = bonobo_config_control_new (NULL);
```

```
pb = bonobo_get_object ("config:eog/display", "Bonobo/PropertyBag", ev);

bonobo_config_control_add_page (config_control, "Title", pb, NULL, NULL);

dialog = bonobo_preferences_new (BONOBO_OBJREF (config_control));
```

The above code creates a full featured preferences dialog with a default layout. You are also able to provide your own layout by passing a callback function to `bonobo_config_control_add_page()`. The following code creates a Page containing a single PropertyEditor widget.

```
create_page_callback_fn (BonoboConfigControl *control,
                 Bonobo_PropertyBag   pb,
                 gpointer             closure,
                 CORBA_Environment   *ev)
{
GtkWidget *w;

w = bonobo_config_widget_new (TC_Bonobo_StorageType, pb, "test-enum",
      NULL, ev);
gtk_widget_show (w);

return w;
}
```

Data Types

We do not make any limitation on data types. Instead BCS allows you to store CORBA:any.

Chapter 2. XMLDB Database Format

Overview

We have decided to use an XML based format because it is human readable any easy to edit. You can use this format for small or medium sized configuration databases. Please keep in mind that the whole XML tree is held in memory, so storing a big database with several megabytes is a bad idea.

XMLDB files are split into sections, and each section contains the entries for a specified directory. The directory name is specified with the path attribute of the section element. The section without a path is the root directory.

```
<?xml version="1.0"?>
<bonobo-conf>
  <section>
    <entry name="value1" type="string" value="a value in the root dir"/>
  </section>

  <section path="subdir">
    <entry name="test-long" type="long" value="5"/>
    <entry name="test-string" type="string" value="test"/>
  </section>

  <section path="doc/subdir">
    <entry name="test-string" type="string" value="a string"/>
  </section>
</bonobo-conf>
```

The <entry> element has several formats. The simplest form is when the type and the value are specified as attributes of the element, as used in the previous example. XMLDB knows the following type string:

Table 2-1. Type attribute values

attrivute value	description
short	16bit signed integer
ushort	16bit unsigned integer
long	32bit signed integer
ulong	32bit unsigned integer

attrivute value	**description**
float	single precision floating point
double	double precision floating point
char	8bit character
string	string
boolean	boolean values

Another form is to use an <value> element. This format is especially useful for localised values. You can use the xml-i18n-tools to generate such files:

```
<entry name="test-string" type="string">
  <value>a string</value>
  <value xml:lang="de">ein string</value>
</entry>
```

The last form is to use a <any> element. This form is used to store complex data types:

```
<entry name="storagetype">
  <any>
    <type name="StorageType" repo_id="IDL:Bonobo/StorageType:1.0"
     tckind="17" length="0" sub_parts="2">
      <subnames>
        <name>STORAGE_TYPE_REGULAR</name>
        <name>STORAGE_TYPE_DIRECTORY</name>
      </subnames>
    </type>
    <value>0</value>
  </any>
</entry>
```

Chapter 3. Function Reference

bonobo-config-database

Name

bonobo-config-database — Configuration Databases

Synopsis

```
#define     BONOBO_CONFIG_DATABASE_TYPE
struct      BonoboConfigDatabasePrivate;
struct      BonoboConfigDatabase;
typedef     BonoboConfigDatabaseClass;
gchar*      bonobo_config_get_string        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
gint16      bonobo_config_get_short         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
guint16     bonobo_config_get_ushort        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
gint32      bonobo_config_get_long          (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
guint32     bonobo_config_get_ulong         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
gfloat      bonobo_config_get_float         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
gdouble     bonobo_config_get_double        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
gboolean    bonobo_config_get_boolean       (Bonobo_ConfigDatabase db,
                                             const char *key,
```

```
                                        CORBA_Environment *opt_ev);
gchar       bonobo_config_get_char      (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         CORBA_Environment *opt_ev);
CORBA_any*  bonobo_config_get_value     (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         CORBA_TypeCode opt_tc,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_string    (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         const char *value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_short     (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         gint16 value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_ushort    (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         guint16 value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_long      (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         gint32 value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_ulong     (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         guint32 value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_float     (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         gfloat value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_double    (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         gdouble value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_boolean   (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         gboolean value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_char      (Bonobo_ConfigDatabase db,
                                         const char *key,
                                         gchar value,
                                         CORBA_Environment *opt_ev);
void        bonobo_config_set_value     (Bonobo_ConfigDatabase db,
```

```
                                        const char *key,
                                        CORBA_any *value,
                                        CORBA_Environment *opt_ev);
```

Description

Configuration databases are the low level API to access configuration data. All other APIs are build on top of this one. There are some handy C wrapper functions in case you don't want to use the CORBA interface. This also provides a smooth migration path if you previously used GConf or the old gnome-config system.

Details

BONOBO_CONFIG_DATABASE_TYPE

```
#define BONOBO_CONFIG_DATABASE_TYPE        (bonobo_config_database_get_type ())
```

struct BonoboConfigDatabasePrivate

```
struct BonoboConfigDatabasePrivate;
```

struct BonoboConfigDatabase

```
struct BonoboConfigDatabase {
BonoboXObject base;

gboolean writeable;

BonoboConfigDatabasePrivate *priv;
};
```

BonoboConfigDatabaseClass

```
typedef struct {
BonoboXObjectClass  parent_class;

POA_Bonobo_ConfigDatabase__epv epv;

        /*
         * virtual methods
         */

CORBA_any      *(*get_value)    (BonoboConfigDatabase *db,
 const CORBA_char     *key,
 const CORBA_char     *locale,
 CORBA_Environment    *ev);

void            (*set_value)    (BonoboConfigDatabase *db,
 const CORBA_char     *key,
 const CORBA_any      *value,
 CORBA_Environment    *ev);

Bonobo_KeyList *(*list_dirs)    (BonoboConfigDatabase *db,
 const CORBA_char     *dir,
 CORBA_Environment    *ev);

Bonobo_KeyList *(*list_keys)    (BonoboConfigDatabase *db,
 const CORBA_char     *dir,
 CORBA_Environment    *ev);

CORBA_boolean   (*dir_exists)   (BonoboConfigDatabase *db,
 const CORBA_char     *dir,
 CORBA_Environment    *ev);

void            (*remove_value) (BonoboConfigDatabase *db,
 const CORBA_char     *key,
 CORBA_Environment    *ev);

void            (*remove_dir)   (BonoboConfigDatabase *db,
 const CORBA_char     *dir,
 CORBA_Environment    *ev);

void            (*sync)         (BonoboConfigDatabase *db,
 CORBA_Environment    *ev);

} BonoboConfigDatabaseClass;
```

bonobo_config_get_string ()

```
gchar*      bonobo_config_get_string        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a string from the configuration database

db : a reference to the database object

key : key of the value to get

opt_ev : an optional CORBA_Environment to return failure codes

Returns : the value contained in the database, or zero on error.

bonobo_config_get_short ()

```
gint16      bonobo_config_get_short         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a 16 bit integer from the configuration database

db : a reference to the database object

key : key of the value to get

opt_ev : an optional CORBA_Environment to return failure codes

Returns : the value contained in the database.

bonobo_config_get_ushort ()

```
guint16     bonobo_config_get_ushort        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a 16 bit unsigned integer from the configuration database

db : a reference to the database object

key : key of the value to get

opt_ev : an optional CORBA_Environment to return failure codes

Returns : the value contained in the database.

bonobo_config_get_long ()

```
gint32      bonobo_config_get_long          (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a 32 bit integer from the configuration database

db : a reference to the database object

key : key of the value to get

opt_ev : an optional CORBA_Environment to return failure codes

Returns : the value contained in the database.

bonobo_config_get_ulong ()

```
guint32     bonobo_config_get_ulong         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a 32 bit unsigned integer from the configuration database

db : a reference to the database object

key : key of the value to get

opt_ev : an optional CORBA_Environment to return failure codes

Returns : the value contained in the database.

bonobo_config_get_float ()

```
gfloat      bonobo_config_get_float         (Bonobo_ConfigDatabase db,
                                             const char *key,
```

```
                                        CORBA_Environment *opt_ev);
```

Get a single precision floating point value from the configuration database

db : a reference to the database object
key : key of the value to get
opt_ev : an optional CORBA_Environment to return failure codes
Returns : the value contained in the database.

bonobo_config_get_double ()

```
gdouble     bonobo_config_get_double        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a double precision floating point value from the configuration database

db : a reference to the database object
key : key of the value to get
opt_ev : an optional CORBA_Environment to return failure codes
Returns : the value contained in the database.

bonobo_config_get_boolean ()

```
gboolean    bonobo_config_get_boolean       (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a boolean value from the configuration database

db : a reference to the database object
key : key of the value to get
opt_ev : an optional CORBA_Environment to return failure codes
Returns : the value contained in the database.

bonobo_config_get_char ()

```
gchar       bonobo_config_get_char          (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_Environment *opt_ev);
```

Get a 8 bit character value from the configuration database

db : a reference to the database object
key : key of the value to get
opt_ev : an optional CORBA_Environment to return failure codes
Returns : the value contained in the database.

bonobo_config_get_value ()

```
CORBA_any*  bonobo_config_get_value         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_TypeCode opt_tc,
                                             CORBA_Environment *opt_ev);
```

Get a value from the configuration database

db : a reference to the database object
key : key of the value to get
opt_tc : the type of the value, optional
opt_ev : an optional CORBA_Environment to return failure codes
Returns : the value contained in the database, or zero on error.

bonobo_config_set_string ()

```
void        bonobo_config_set_string        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             const char *value,
                                             CORBA_Environment *opt_ev);
```

Set a string value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_short ()

```
void        bonobo_config_set_short         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             gint16 value,
                                             CORBA_Environment *opt_ev);
```

Set a 16 bit integer value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_ushort ()

```
void        bonobo_config_set_ushort        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             guint16 value,
                                             CORBA_Environment *opt_ev);
```

Set a 16 bit unsigned integer value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_long ()

```
void        bonobo_config_set_long          (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             gint32 value,
                                             CORBA_Environment *opt_ev);
```

Set a 32 bit integer value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_ulong ()

```
void        bonobo_config_set_ulong         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             guint32 value,
                                             CORBA_Environment *opt_ev);
```

Set a 32 bit unsigned integer value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_float ()

```
void        bonobo_config_set_float         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             gfloat value,
                                             CORBA_Environment *opt_ev);
```

Set a single precision floating point value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_double ()

```
void        bonobo_config_set_double        (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             gdouble value,
                                             CORBA_Environment *opt_ev);
```

Set a double precision floating point value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_boolean ()

```
void        bonobo_config_set_boolean       (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             gboolean value,
                                             CORBA_Environment *opt_ev);
```

Set a boolean value in the configuration database.

db : a reference to the database object
key : key of the value to set
value : the new value
opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_char ()

```
void        bonobo_config_set_char          (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             gchar value,
                                             CORBA_Environment *opt_ev);
```

Set a 8 bit characte value in the configuration database.

db : a reference to the database object

key : key of the value to set

value : the new value

opt_ev : an optional CORBA_Environment to return failure codes

bonobo_config_set_value ()

```
void        bonobo_config_set_value         (Bonobo_ConfigDatabase db,
                                             const char *key,
                                             CORBA_any *value,
                                             CORBA_Environment *opt_ev);
```

Set a value in the configuration database.

db : a reference to the database object

key : key of the value to set

value : the new value

opt_ev : an optional CORBA_Environment to return failure codes

bonobo-config-utils

Name

bonobo-config-utils — Utility functions

Synopsis

```
BonoboUINode* bonobo_config_xml_encode_any  (const CORBA_any *any,
                                             const char *name,
                                             CORBA_Environment *ev);
CORBA_any*  bonobo_config_xml_decode_any    (BonoboUINode *node,
                                             const char *locale,
                                             CORBA_Environment *ev);
char*       bonobo_config_any_to_string     (CORBA_any *any);
char*       bonobo_config_dir_name          (const char *key);
char*       bonobo_config_leaf_name         (const char *key);
```

Description

This section describes a number of utility functions.

Details

bonobo_config_xml_encode_any ()

```
BonoboUINode* bonobo_config_xml_encode_any  (const CORBA_any *any,
                                             const char *name,
                                             CORBA_Environment *ev);
```

This routine encodes *any* into a XML tree.

any : the Any to encode
name : the name of the entry

ev : a corba exception environment

Returns : the XML tree representing the Any

bonobo_config_xml_decode_any ()

```
CORBA_any*  bonobo_config_xml_decode_any    (BonoboUINode *node,
                                             const char *locale,
                                             CORBA_Environment *ev);
```

This routine is the converse of bonobo_config_xml_encode_any. It hydrates a serialized CORBA_any.

node : the parsed XML representation of an any

locale : an optional locale

ev : a corba exception environment

Returns : the CORBA_any or NULL on error

bonobo_config_any_to_string ()

```
char*       bonobo_config_any_to_string     (CORBA_any *any);
```

Converts the *any* into a string representation.

any : the Any to encode

Returns : a string representation of the any.

bonobo_config_dir_name ()

```
char*       bonobo_config_dir_name          (const char *key);
```

Gets the directory components of a configuration key. If *key* does not contain a directory component it returns `NULL`.

key : a configuration key

Returns : the directory components of a configuration key.

bonobo_config_leaf_name ()

```
char*       bonobo_config_leaf_name         (const char *key);
```

Gets the name of a configuration key without the leading directory component.

key : a configuration key

Returns : the name of a configuration key without the leading directory component.

bonobo-property-editor

Name

bonobo-property-editor — A Class to support the writing of ProperyEditors.

Synopsis

```
#define     BONOBO_PROPERTY_EDITOR_PREFIX
#define     BONOBO_PROPERTY_EDITOR_TYPE
struct      BonoboPropertyEditorPrivate;
typedef     BonoboPropertyEditor;
void        (*BonoboPropertyEditorSetFn)    (BonoboPropertyEditor *editor,
                                             BonoboArg *value,
                                             CORBA_Environment *ev);
typedef     BonoboPropertyEditorClass;
BonoboPropertyEditor* bonobo_property_editor_new
                                            (GtkWidget *widget,
                                             BonoboPropertyEdi-
torSetFn set_cb);
```

```
void        bonobo_property_editor_set_value
                                            (BonoboPropertyEditor *editor,
                                             const BonoboArg *value,
                                             CORBA_Environment *opt_ev);
BonoboObject* bonobo_property_editor_basic_new
                                            (CORBA_TypeCode tc);
BonoboObject* bonobo_property_editor_boolean_new
                                            ();
BonoboObject* bonobo_property_editor_enum_new
                                            ();
BonoboObject* bonobo_property_editor_default_new
                                            ();
Bonobo_PropertyEditor bonobo_property_editor_resolve
                                            (CORBA_TypeCode tc,
                                             CORBA_Environment *ev);
```

Description

Details

BONOBO_PROPERTY_EDITOR_PREFIX

```
#define BONOBO_PROPERTY_EDITOR_PREFIX "OAFIID:Bonobo_PropertyEditor_"
```

BONOBO_PROPERTY_EDITOR_TYPE

```
#define BONOBO_PROPERTY_EDITOR_TYPE        (bonobo_property_editor_get_type ())
```

struct BonoboPropertyEditorPrivate

```
struct BonoboPropertyEditorPrivate;
```

BonoboPropertyEditor

```
typedef struct {
BonoboControl base;

CORBA_TypeCode tc; /* read only */

BonoboPropertyEditorPrivate *priv;
} BonoboPropertyEditor;
```

BonoboPropertyEditorSetFn ()

```
void        (*BonoboPropertyEditorSetFn)    (BonoboPropertyEditor *editor,
                                             BonoboArg *value,
                                             CORBA_Environment *ev);
```

editor :

value :

ev :

BonoboPropertyEditorClass

```
typedef struct {
BonoboControlClass parent_class;

POA_Bonobo_PropertyEditor__epv epv;

/* virtual methods */

BonoboPropertyEditorSetFn set_value;

Bonobo_PropertyEditor_PropertyEditorSet *
(*get_sub_editors)   (BonoboPropertyEditor  *editor,
      CORBA_Environment     *ev);

} BonoboPropertyEditorClass;
```

bonobo_property_editor_new ()

```
BonoboPropertyEditor* bonobo_property_editor_new
                                        (GtkWidget *widget,
                                         BonoboPropertyEdi-
torSetFn set_cb);
```

widget :

set_cb :

Returns :

bonobo_property_editor_set_value ()

```
void        bonobo_property_editor_set_value
                                        (BonoboPropertyEditor *editor,
                                         const BonoboArg *value,
                                         CORBA_Environment *opt_ev);
```

editor :

value :

opt_ev :

bonobo_property_editor_basic_new ()

```
BonoboObject* bonobo_property_editor_basic_new
                                        (CORBA_TypeCode tc);
```

tc :

Returns :

bonobo_property_editor_boolean_new ()

```
BonoboObject* bonobo_property_editor_boolean_new
```

```
                                        ();
```

Returns :

bonobo_property_editor_enum_new ()

```
BonoboObject* bonobo_property_editor_enum_new
                                        ();
```

Returns :

bonobo_property_editor_default_new ()

```
BonoboObject* bonobo_property_editor_default_new
                                        ();
```

Returns :

bonobo_property_editor_resolve ()

```
Bonobo_PropertyEditor bonobo_property_editor_resolve
                                        (CORBA_TypeCode tc,
                                         CORBA_Environment *ev);
```

tc :

ev :

Returns :