TANGO - DBAPI (V1.0)

A.Götz 31 October 2000

This paper documents the C++ API for the TANGO 2 database.

Contents

1 Introduction

The TANGO database is implemented today as a TANGO device server. To access it the user has the CORBA interface cmd_inout(). This expects and returns all parameters as ascii strings thereby making the database laborious to use for retrieving device properties and information. In order to simplify this access a high-level api has been implemented which hides the low-level formatting necessary to convert the cmd_inout() return values into binary values and all CORBA aspects of the TANGO. All data types are native C++ data types e.g. simple type or vectors. This note documents this high level interface for C++.

2 Basic Philosophy

The basic philosophy is to have high level classes for the database, properties, device and class info in the database and a class for sending and receiving one or more database values. The high level classes are Tango::Database, Tango::DbDevice, Tango::DbClass, Tango::DbServer. All classes and data types are defined as part of the Tango namespace.

3 Getting started

The quickest way of getting started is by studying this example :

```
/*
 * example of a client using the TANGO database api.
 */
#include <tango_ds.h>
#include <dbapi.h>
using namespace Tango;
main(unsigned int argc, char **argv)
{
        Database *dbase = new Database();
        DbDatum sl_parity("parity"), sl_baudrate("baudrate"), sl_stopbits("stopb its");
        DbData sl_props_in, sl_props_out;
        DbDevInfo dev_info;
        DbDevExportInfo export_info;
        DbDevImportInfo import_info;
        vector<short> stopbits;
        short baudrate=19200;
        string parity("even");
        string device_name("my/serial/device");
         try {
//
// get general info about the database
//
                cout << "dbase->get_info() : " << dbase->get_info() << endl;
//
// update some device properties in the database
//
                sl_parity << parity;
                sl_baudrate << baudrate;
```

```
                stopbits.resize(3);
                stopbits[0] = 0;
                stopbits[1] = 1;
                stopbits[2] = 2;
                sl_stopbits << stopbits;
                sl_props_in.push_back(sl_parity);
                sl_props_in.push_back(sl_baudrate);
                sl_props_in.push_back(sl_stopbits);
                cout << "dbase->put_device_property() called" << endl;
                dbase->put_device_property(device_name, sl_props_in);
//
// query the database for some device properties
//
                sl_props_out.push_back(DbDatum("parity"));
                sl_props_out.push_back(DbDatum("baudrate"));
                sl_props_out.push_back(DbDatum("stopbits"));
                cout << "dbase->put_device_property(" << device_name << ") called" << endl;
                dbase->get_device_property(device_name, sl_props_out);
                cout << device_name;
                cout << " parity = " << sl_props_out[0].value_string[0] << ",";
                sl_props_out[1] >> baudrate;
                cout << " baudrate = " << baudrate << ", ";
                sl_props_out[2] >> stopbits;
                cout << " stopbits = " << stopbits.size() << endl;
//
// add a device to the database
//
                dev_info.name = device_name;
                dev_info.class_ = "Test";
                dev_info.server = "serial/test";
                cout << "dbase->add_device(" << device_name << ") called" << endl;
                dbase->add_device(dev_info);
//
// export a device in the database
//
                export_info.name = device_name;
                export_info.server = "andy/test";
                export_info.ior = "ior";
                export_info.host = "dumela";
                export_info.version = "1.0";
                export_info.pid = getpid();
                cout << "dbase->export_device(" << device_name << ") called" <<endl;
                dbase->export_device(export_info);
//
// import a device from the database
//
                cout << "dbase->import_device(" << device_name << ") called" << endl;
                import_info = dbase->import_device(device_name);
                cout << "info : name " << import_info.name << ", ";
                cout << "exported " << import_info.exported << ", ";
```

```
                cout << "ior " << import_info.ior << ", ";
                cout << "version " << import_info.version << ", ";
                cout << endl;
        }
        catch (DevFailed &e)
        {
                Util::print_exception(e);
                exit(-1);
        }
}
```

Modify this example to fit your device server or client's needs, compile it and link with the library -ldbapi. Run it and forget about those painful early Tango days when you had to learn CORBA and manipulate Any's. Life's going to easy and fun from now on !

4 Tango::Database

A highlevel object which contains the link to the database. It has methods for all database commands e.g. get_device_property(), device_list(), info(), etc.

4.1 Database::Database()

Create a TANGO Database object. The constructor uses the environment variable "TANGO_HOST" to determine which instance of the TANGO database to connect to. Example :

```
using namespace Tango;
Database *db = new Database();
```

4.2 Database::get_info()

Query the database for some general info about the tables in the database. Result is returned as a string. Example :

```
cout << db->get_info() << endl;
```

4.3 Database::add_device(DbDevInfo&)

Add a device to the database. The device name, server and class are specified in the DbDevInfo structure. Example :

```
DbDevInfo my_device_info;
my_device_info.name = "my/own/device";
my_device_info._class = "MyDevice";
my_device_info.server = "MyServer/test";
db->add_device(my_device_info);
```

4.4 Database::delete_device(string)

Delete the device of the specified name from the database. Example

```
db->delete_device("my/own/device");
```

4.5 Database::import_device(string)

Query the database for the export info of the specified device. The command returns the information in a DbDevExportInfo structure. Example :

```
DbDevImportInfo my_device_import;
my_device_import = db->import_device("my/own/device");
cout << " device " << my_device_import.name;
cout << "exported " << my_device_import.exported;
cout << "ior " << my_device_import.ior;
cout << "version " << my_device_import.version;
cout << endl;
```

4.6 Database::export_device(DbDevExportInfo&)

Update the export info for this device in the database. Device name, server, class, pid and version are specified in the DbDevExportInfo structure. Example :

```
DbDevExportInfo my_device_export;
my_device_export.name = "my/own/device";
my_device_export.server = "MyServer/test";
my_device_export.ior = "the real ior";
my_device_export.host = "dumela";
my_device_export.version = "1.0";
my_device_export.pid = get_pid();
db->export_device(my_device_export);
```

4.7 Database::unexport_device(string)

Mark the specified device as unexported in the database. Example :

```
db->unexport_device("my/own/device");
```

4.8 Database::add_server(DbDevInfos&)

Add a group of devices to the database. The device names, server names and classes are specified in the vector of DbDevInfo structures.

4.9 Database::delete_server(string)

Delete the device server and its associated devices from the database.

4.10 Database::export_server(DbDevExportInfos&)

Export a group of devices to the database. The device names, IOR, class, server name, pid etc. are specified in the vector of DbDevExportInfo structures.

4.11 Database::unexport_server(string)

Mark all devices exported for this server as unexported.

4.12 Database::get_device_name(string, string)

Query the database for a list of devices served by the specified server (1st parameter) and of the specified class (2nd parameter). The method returns a DbDatum type. The device names are stored as an array of strings. Here is two code example of how to extract the names from the DbDatum type :

```
vector<string> device_names;
device_names << db_datum;
```

or :

```
for (int i=0; i< db_datum.size(); i++)
{
        device_name[i] = db_datum.value_string[i];
}
```

4.13 Database::get_device_alias(string)

Query the database for a list of aliases for the specified device. This method returns a DbDatum type. See the method get_device_name() for an example of how to extract the list of aliases from the DbDatum type.

4.14 Database::get_device_domain(string)

Query the database for a list of device domain names which match the wildcard provided. Wildcard character is * and matches any number of characters. Domain names are case insensitive. This method returns a DbDatum type. See the method get_device_name() for an example of how to extract the list of aliases from the DbDatum type.

4.15 Database::get_device_family(string)

Query the database for a list of device family names which match the wildcard provided. Wildcard character is * and matches any number of characters. Family names are case insensitive. This method returns a DbDatum type. See the method get_device_name() for an example of how to extract the list of aliases from the DbDatum type.

4.16 Database::get_device_member(string)

Query the database for a list of device member names which match the wildcard provided. Wildcard characters is * and matches any number of characters. Member names are case insensitive. This method returns a DbDatum type. See the method get_device_name() for an example of how to extract the list of aliases from the DbDatum type.

4.17 Database::get_property(string, DbData&)

Query the database for a list of object (i.e. non-device) properties for the specified object. The property names are specified by the vector of DbDatum structures. The method returns the properties in the same DbDatum structures. To retrieve the properties use the extract operator >>. Here is an example of how to use the DbData type to specify and extract properties :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->get_property("mymotor", db_data);
float velocity, acceleration;
db_data[0] >> velocity;
db_data[1] >> acceleration;
```

4.18 Database::put_property(string, DbData&)

Insert or update a list of properties for the specified object. The property names and their values are specified by the vector of DbDatum structures. Use the insert operator >> to insert the properties into the DbDatum structures. Here is an example of how to insert properties into the database using this method :

```
DbDatum velocity("velocity"), acceleration("acceleration");
DbData db_data;
velocity << 100000.0;
acceleration << 500000.0;
db_data.push_back(velocity);
db_data.push_back(acceleration);
db->put_property("mymotor", db_data);
```

4.19 Database::delete_property(string, DbData&)

Delete a list of properties for the specified object. The property names are specified by the vector of DbDatum structures. Here is an example of how to delete properties from the database using this method :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->delete_property("mymotor", db_data);
```

4.20 Database::get_device_property(string, DbData&)

Query the database for a list of device properties for the specified object. The property names are specified by the vector of DbDatum structures. The method returns the properties in the same DbDatum structures. To retrieve the properties use the extract operator >>. Here is an example of how to use the DbData type to specify and extract properties :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->get_device_property("id11/motor/1", db_data);
float velocity, acceleration;
db_data[0] >> velocity;
db_data[1] >> acceleration;
```

4.21 Database::put_device_property(string, DbData&)

Insert or update a list of properties for the specified device. The property names and their values are specified by the vector of DbDatum structures. Use the insert operator >> to insert the properties into the DbDatum structures. Here is an example of how to insert properties into the database using this method :

```
DbDatum velocity("velocity"), acceleration("acceleration");
DbData db_data;
velocity << 100000.0;
acceleration << 500000.0;
db_data.push_back(velocity);
db_data.push_back(acceleration);
db->put_device_property("id11/motor/1", db_data);
```

4.22 Database::delete_device_property(string, DbData&)

Delete a list of properties for the specified device. The property names are specified by the vector of DbDatum structures. Here is an example of how to delete properties from the database using this method :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->delete_device_property("id11/motor/1", db_data);
```

4.23 Database::get_device_attribute_property(string, DbData&)

Query the database for a list of device attribute properties for the specified object. The attribute names are specified by the vector of DbDatum structures. The method returns all the properties for the specified attributes. The attribute names are returned with the number of properties specified as their value. To retrieve the properties use the extract operator >>. Here is an example of how to use the DbData type to specify and extract attribute properties :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->get_device_attribute_property("id11/motor/1", db_data);
float vel_max, vel_min, acc_max, acc_min;
int n_vel_props, n_acc_props, index;
index = 0;
db_data[0] >> n_vel_props;
```

```
db_data[n_vel_props+1] >> n_acc_props;
for (int i=1; i< n_vel_props; i++)
{
        if (db_data[i].name == "min") db_data[i] >> vel_min;
       else if (db_data[i].name == "max") db_data[i] >> vel_max;
}
for (int i=n_vel_props+2; i< n_acc_props; i++)
{
        if (db_data[i].name == "min") db_data[i] >> acc_min;
        else if (db_data[i].name == "max") db_data[i] >> acc_max;
}
```

4.24 Database::put_device_attribute_property(string, DbData&)

Insert or update a list of attribute properties for the specified device. The attribute property names and their values are specified by the vector of DbDatum structures. Use the insert operator >> to insert the properties into the DbDatum structures. Here is an example of how to insert properties into the database using this method :

```
DbDatum velocity("velocity"), vel_min("min"), vel_max("max");
DbDatum acceleration("acceleration"), acc_min("min"), acc_max("max");
DbData db_data;
velocity << 2;
vel_min << 0.0;
vel_max << 1000000.0;
db_data.push_back(velocity);
db_data.push_back(vel_min);
db_data.push_back(vel_max);
acceleration << 2;
acc_min << 0.0;
acc_max << 8000000;
db_data.push_back(acceleration);
db_data.push_back(acc_min);
db_data.push_back(acc_max);
db->put_device_attribute_property("id11/motor/1", db_data);
```

4.25 Database::delete_device_attribute_property(string, DbData&)

Delete a list of attribute properties for the specified device. The attribute names are specified by the vector of DbDatum structures. All properties belonging to the listed attributes are deleted. Here is an example of how to delete properties from the database using this method :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->delete_device_attribute_property("id11/motor/1", db_data);
```

4.26 Database::get_class_property(string, DbData&)

Query the database for a list of class properties. The property names are specified by the vector of DbDatum structures. The method returns the properties in the same DbDatum structures. To retrieve the properties use the extract operator >>. Here is an example of how to use the DbData type to specify and extract properties :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->get_class_property("StepperMotor", db_data);
float velocity, acceleration;
db_data[0] >> velocity;
db_data[1] >> acceleration;
```

4.27 Database::put_class_property(string, DbData&)

Insert or update a list of properties for the specified class. The property names and their values are specified by the vector of DbDatum structures. Use the insert operator >> to insert the properties into the DbDatum structures. Here is an example of how to insert properties into the database using this method :

```
DbDatum velocity("velocity"), acceleration("acceleration");
DbData db_data;
velocity << 100000.0;
acceleration << 500000.0;
db_data.push_back(velocity);
db_data.push_back(acceleration);
db->put_class_property("StepperMotor", db_data);
```

4.28 Database::delete_class_property(string, DbData&)

Delete a list of properties for the specified class. The property names are specified by the vector of DbDatum structures. Here is an example of how to delete properties from the database using this method :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->delete_class_property("StepperMotor", db_data);
```

4.29 Database::get_class_attribute_property(string, DbData&)

Query the database for a list of class attribute properties for the specified object. The attribute names are specified by the vector of DbDatum structures. The method returns all the properties for the specified attributes. The attribute names are returned with the number of properties specified as their value. To retrieve the properties use the extract operator >>. Here is an example of how to use the DbData type to specify and extract attribute properties :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->get_class_attribute_property("StepperMotor", db_data);
float vel_max, vel_min, acc_max, acc_min;
int n_vel_props, n_acc_props, index;
index = 0;
db_data[0] >> n_vel_props;
db_data[n_vel_props+1] >> n_acc_props;
for (int i=1; i< n_vel_props; i++)
{
        if (db_data[i].name == "min") db_data[i] >> vel_min;
       else if (db_data[i].name == "max") db_data[i] >> vel_max;
}
for (int i=n_vel_props+2; i< n_acc_props; i++)
{
        if (db_data[i].name == "min") db_data[i] >> acc_min;
        else if (db_data[i].name == "max") db_data[i] >> acc_max;
}
```

4.30 Database::put_class_attribute_property(string, DbData&)

Insert or update a list of attribute properties for the specified class. The attribute property names and their values are specified by the vector of DbDatum structures. Use the insert operator >> to insert the properties into the DbDatum structures. Here is an example of how to insert properties into the database using this method :

```
DbDatum velocity("velocity"), vel_min("min"), vel_max("max");
DbDatum acceleration("acceleration"), acc_min("min"), acc_max("max");
DbData db_data;
velocity << 2;
vel_min << 0.0;
vel_max << 1000000.0;
db_data.push_back(velocity);
db_data.push_back(vel_min);
db_data.push_back(vel_max);
acceleration << 2;
acc_min << 0.0;
acc_max << 8000000;
db_data.push_back(acceleration);
db_data.push_back(acc_min);
db_data.push_back(acc_max);
db->put_class_attribute_property("StepperMotor", db_data);
```

4.31 Database::delete_class_attribute_property(string, DbData&)

Delete a list of attribute properties for the specified class. The attribute names are specified by the vector of DbDatum structures. All properties belonging to the listed attributes are deleted. Here is an example of how to delete properties from the database using this method :

```
DbData db_data;
db_data.push_back(DbDatum("velocity"));
db_data.push_back(DbDatum("acceleration"));
db->delete_class_attribute_property("StepperMotor", db_data);
```

5 Tango::DbDevice

A database object for a device which can be used to query or modify properties, import and export information for a device. This class provides an easy to use interface for device objects in the database. It uses the methods of the Database class therefore the reader is referred to these for the exact calling syntax and examples. The following methods are defined for the DbDevice class :

5.1 DbDevice::DbDevice(string)

A constructor for a DbDevice object for a device in the TANGO database specified by the TANGO_HOST environment variable. This method creates a new Database object per DbDevice created.

5.2 DbDevice::DbDevice(string, Database *)

A constructor for a DbDevice object for the device in the specified database. This method reuses the Database supplied by the programmer.

5.3 DbDevice::import_device()

Query the database for the import info of this device. Returns a DbDevImportInfo structure.

5.4 DbDevice::export_device(DbDevExportInfo&)

Update the export info for this device in the database.

5.5 DbDevice::add_device(DbDevInfo&)

Add/Update this device to the database.

5.6 DbDevice::delete_device()

Delete this device from the database.

5.7 DbDevice::get_property(DbData&)

Query the database for the list of properties of this device. See Database::get_device_property() for an example of how to specify and retrieve the properties.

5.8 DbDevice::put_property(DbData&)

Update the list of properties for this device in the database. See Database::put_device_property() for an example of how to specify the properties.

5.9 DbDevice::delete_property(DbData&)

Delete the list of specified properties for this device in the database. See Database::delete_property() for an example of how to specify the properties.

5.10 DbDevice::get_attribute_property(DbData&)

Query the database for the list of attribute properties of this device. See Database::get_device_attribute_property() for an example of how to specify and retrieve the properties.

5.11 DbDevice::put_attribute_property(DbData&)

Update the list of attribute properties for this device in the database. See Database::put_device_attribute_property() for an example of how to specify the properties.

5.12 DbDevice::delete_attribute_property(DbData&)

Delete all properties for the list of specified attributes for this device in the database. See Database::delete_device_attribute_property() for an example of how to specify the properties.

6 Tango::DbClass

A database object for a class which can be used to query or modify class properties.

6.1 DbClass::DbClass(string)

A constructor for a DbClass object for aclass in the TANGO database specified by the TANGO_HOST environment variable. This method creates a new Database object per DbClass created.

6.2 DbClass::DbClass(string, Database *)

A constructor for a DbClass object for the class in the specified database. This method reuses the Database supplied by the programmer.

6.3 DbClass::get_property(DbData&)

Query the database for the list of properties of this class. See Database::get_class_property() for an example of how to specify and retrieve the properties.

6.4 DbClass::put_property(DbData&)

Update the list of properties for this class in the database. See Database::put_class_property() for an example of how to specify the properties.

6.5 DbClass::delete_property(DbData&)

Delete the list of specified properties for this class in the database. See Database::delete_property() for an example of how to specify the properties.

6.6 DbClass::get_attribute_property(DbData&)

Query the database for the list of attribute properties of this class. See Database::get_class_attribute_property() for an example of how to specify and retrieve the properties.

6.7 DbClass::put_attribute_property(DbData&)

Update the list of attribute properties for this class in the database. See Database::put_class_attribute_property() for an example of how to specify the properties.

6.8 DbClass::delete_attribute_property(DbData&)

Delete all properties for the list of specified attributes for this class in the database. See Database::delete_class_attribute_property() for an example of how to specify the properties.

7 Tango::DbDatum

A single database value which has a name, type, address and value and methods for inserting and extracting C++ native types. This is the fundamental type for specifying database properties. Every property has a name and has one or more values associated with it. The values can be inserted and extracted using the operators << and >> respectively. The insert and extract operators are specified for the following C++ types :

1. short
2. unsigned short
3. long
4. unsigned long
5. int
6. float
7. double
8. string
9. vector<string>
10. vector<short>
11. vector<unsigned short>
12. vector<long>
13. vector<unsigned long>

14. vector<float>
15. vector<double>

Here is an example of creating, inserting and extracting some DbDatum types :

```
DbDatum my_short("my_short"), my_long("my_long"), my_string("my_string");
DbDatum my_float_vector("my_float_vector"), my_double_vector("my_double_vector");
string a_string;
short a_short;
long a_long;
vector<float> a_float_vector;
vector<double> a_double_vector;
my_short << 100; // insert a short
my_short >> a_short; // extract a short
my_long << 1000; // insert a long
my_long >> a_long; // extract a long
my_string << string("estas lista a bailar el tango ?"); // insert a string
my_string >> a_string; // extract a string
my_float_vector << a_float_vector // insert a vector of floats
my_float_vector >> a_float_vector; // extract a vector of floats
my_double_vector << a_double_vector; // insert a vector of doubles
my_double_vector >> a_double_vector; // extract a vector of doubles
```

8 Tango::DbData

A vector of Tango::DbDatum structures. DbData is used to send or return one or more database properties or information. It is the standard input and output type for all methods which query and/or update properties in the database.