Guide for the development of a SALOME module in Python

The purpose of this document is to describe briefly the different steps in the development of a SALOME module in Python.

Steps in construction of the example module

The example module chosen to illustrate the process to construct a module is extremely simple. It will contain a single component and this component will have a single service called getBanner that will accept a character string as the sole argument and that will return a character string obtained by concatenation of "Hello" and the input chain. This component will be completed by a graphic GUI written in PyQt.

The different steps in the development will be as follows:

- create a module tree structure
- create a SALOME component that can be loaded by a Python container
- configure the module so that the component is known to SALOME
- add a graphic GUI

Create the module tree structure

Firstly, we will simply put a SALOME component written in Python and that can be loaded by a Python container, into the example module. An idl interface and a Python implementation of the component will be all that are necessary. The following file structure is necessary so that this can be implemented in a SALOME module:

```
+ PYHELLO1_SRC
  + build_configure
  + configure.ac
  + Makefile.am
  + adm_local
    + Makefile.am
    + unix
      + Makefile.am
      + make_common_starter.am
      + config_files
        + Makefile.am
        + check_PYHELLO.m4
  + bin
    + Makefile.am
    + VERSION.in
    + runAppli.in
    + myrunSalome.py
  + idl
```

```
  + Makefile.am
  + PYHELLO_Gen.idl
+ src
  + Makefile.am
  + PYHELLO
    + Makefile.am
    + PYHELLO.py
+ doc
```

The module name is PYHELLO, the component name is PYHELLO and all the files will be put in a directory named PYHELLO1_SRC. All files are essential except for VERSION.in, runAppli.in and runSalome.py. VERSION.in is used to document the module, it must give its version and its compatibilities or incompatibilities with other modules. Therefore, it is strongly recommended but is not essential for operation. The runAppli.in and runSalome.py files are not essential but make the example easier to use.

Warning: the files of the basic platform (KERNEL) must not be copied to initialise a module tree structure. It is usually preferable to copy files from another module such as GEOM or MED.

Implementation of automake, configure

SALOME uses autoconf and automake to build the configure script that is used for the installation to test the system configuration and to preconfigure the module construction Makefile files. The build_configure file contains a procedure that starts from configure.ac and uses automake to build the configure script. automake starts from Makefile.am files to build Makefile.in files. All files with an "in" extension are skeletons that will be transformed by the configure process.

Almost all files used for this process are located in the basic platform that is referenced by the KERNEL_ROOT_DIR environment variable as well as GUI_ROOT_DIR for the graphical user interface (GUI). However, some files must be modified as a function of the target module. This is the case for build_configure and configure.ac files that usually need to be adapted.

The basic files for configuration of the KERNEL module and other modules are collected in the salome_adm directory of the KERNEL module. However, in order to be able to use the CORBA objects of the KERNEL module, the files in the salome_adm directory have to be overwritten, using the make_common_starter.am file in the adm_local directory of the example module.

config_files is a directory in which the m4 files that are used to test the configuration of the system in the configure process can be placed. If the salome_adm files are not sufficient, others can be added in adm_local.

The idl directory

The idl directory requires a Makefile.am that must make the compilation of the idl PYHELLO_Gen.idl file and install all the generated files in the right module installation directories. The BASE_IDL_FILES target has to be modified to reach this goal.

The idl file itself must define a CORBA module for which the name must be different from the module name to avoid name conflicts and define a CORBA interface that is derived at least from the Component interface of the Engines module. The name of the CORBA module will be PYHELLO_ORB and the name of the interface will be PYHELLO_Gen.

The src directory

The src directory will contain all components and the GUI for the module. Each of these entities must have its own directory.

For the moment, the module will only contain a single directory for the engine of the PYHELLO component and its name will be PYHELLO.

The Makefile.am will simply trigger the path of sub-directories described by the SUBDIRS target.

The PYHELLO directory

This directory contains the Python module that represents the component and therefore contains the PYHELLO class and a Makefile.am file that simply exports the PYHELLO.py module into the installation directory of the SALOME module.

The PYHELLO.py module contains the PYHELLO class that is derived from the PYHELLO_Gen interface of the CORBA PYHELLO_ORB_POA module and the SALOME_ComponentPy_i class of the SALOME_ComponentPy module.

The doc directory

This contains nothing for the moment. It could contain this document.

The bin directory

VERSION.in is used to document the module, it must define its version and its compatibilities or incompatibilities with other modules. Therefore, it is strongly recommended but is not essential for operation.

The runAppli.in file is the equivalent of the runSalome in the KERNEL module configured to implement the KERNEL module and this PYHELLO module.

The myrunSalome.py file is the file of the KERNEL module modified to run only with a Python container, with the test function that creates the PYHELLO component instead of a MED component, and automatic completion in Python.

Creating a component that can be loaded by a container

The files presented above are sufficient to build and install the PYHELLO1_SRC module, to start the SALOME platform composed of the KERNEL and PYHELLO1 modules, and to request the Python container to load a PYHELLO component.

All the following steps are only possible if the SALOME prerequisite software is accessible in the module developer environment.

Construction, installation

In PYHELLO1_SRC, enter:

```
export KERNEL_ROOT_DIR=<KERNEL installation path>
./build_configure
```

Go into ../PYHELLO1_BUILD and enter:

```
../PYHELLO1_SRC/configure --prefix=<PYHELLO1 installation path>
make
make install
```

Running the platform

Move into the <PYHELLO1 module installation path> and enter:

```
./bin/salome/runAppli
```

This command runs SALOME configured for KERNEL and the PYHELLO1 module. At the end of running, the user sees a Python interpreter configured for SALOME that provides access to SALOME CORBA objects.

runAppli is a shell that executes a Python script, by passing arguments to it in a command line:

```
python -i $PYHELLO_ROOT_DIR/bin/salome/myrunSalome.py --modules=PYHELLO --killall
```

These arguments state that the myrunSalome.py script located in the PYHELLO module will be used, that the PYHELLO component will be activated and all SALOME processes that existed before the run will be killed.

This command will not function unless the following environment variables have previously been set:

```
export KERNEL_ROOT_DIR=<KERNEL installation path>
export PYHELLO_ROOT_DIR=<PYHELLO installation path>
```

Warning: it is possible that the SALOME run will not reach the end. In some circumstances, the time to start CORBA servers may be long and could exceed the timeout. If the reason for this is that the time to load dynamic libraries is long, it is possible that a second run immediately afterwards will be successful.

Loading the example component

The PYHELLO_ORB module has to be imported before making a request to load the component into the Python container, to obtain access to methods of the component. This Python container was made accessible in the runSalome.py by means of the container variable:

```
import PYHELLO_ORB
c=container.load_impl("PYHELLO","PYHELLO")
c.makeBanner("Christian")
```

The last instruction must return 'Hello Christian'.

Proceed as follows to see CORBA objects created by these actions:

```
clt.showNS()
```

Declared SALOME component

For the moment, the PYHELLO component was loaded by making a direct request to the Python container. This is not the standard method for loading a component. The normal method uses the LifeCycle service that uses catalog services to identify the component and its properties and then calls the requested container to load the component.

Before this method can be used, the component must be declared in a catalog in the XML format, for which the name must be <Module>Catalog.xml. In our case, it will be PYHELLOCatalog.xml. This catalog will be stored in the resources directory.

Updated tree structure:

```
+ PYHELLO1_SRC
  + build_configure
  + configure.ac
  + Makefile.am
  + adm_local
  + bin
  + idl
  + src
  + doc
  + resources
    + PYHELLOCatalog.xml
```

The remainder of the files are identical, apart from adding the resources directory and the PYHELLOCatalog.xml file. However, the Makefile.am has to be modified so that the catalog is actually installed in the installation directory. It simply needs to be specified in the salomeres_SCRIPTS target.

Construction, installation

There is no need to do another configure to take account of this modification. All that is necessary is to enter PYHELLO1_BUILD and then:

```
./config.status
make
make install
```

Starting the platform

The platform is started in the same way as before. Go into PYHELLO1_INSTALL and do:

```
./bin/salome/runAppli
```

Loading the example component

The method of loading the component is not very different from that described above. The services of the LifeCycle module are used in this case instead of calling the container directly. The call sequence is contained in the runSalome.Py test function.

```
c=test(clt)
c.makeBanner("Christian")
```

The test function creates the LifeCycle. It then asks for the PYHELLO component to be loaded in the FactoryServerPy container:

```
def test(clt):
     """
      Test function that creates an instance of PYHELLO component
      usage : pyhello=test(clt)
     """
     import LifeCycleCORBA
     lcc = LifeCycleCORBA.LifeCycleCORBA(clt.orb)
     import PYHELLO_ORB
     pyhello = lcc.FindOrLoadComponent("FactoryServerPy", "PYHELLO")
     return pyhello
```

Loading from the application interface (IAPP)

Before a component can be loaded dynamically using the IAPP components bar, the icon representing the component will have to be declared in the catalog. It is declared by simply adding a line for the icon to the component catalog:

```
<component-icon>PYHELLO.png</component-icon>
```

and putting the corresponding file in the module resources directory.

Adding a graphic GUI

The next step to complete the module consists of adding a graphic interface to the PYHELLO component, that will be written in Python using the Qt widgets library. This graphic interface must be integrated into the SALOME application interface (IAPP), and therefore must respect some constraints that we will see.

Firstly note the contour of the GUI of a component. The behaviour of the GUI is given by a Python module that has a standard name <Module>GUI.py. It must propose conventional entry points that the IAPP will use to activate this GUI or to inform it of specific events. GUI commands are activated through a menu bar and a button bar that are integrated into the menu bar and into the IAPP button bar.

Python module implanting the behaviour of the GUI

The behaviour of the PYHELLO component GUI is implanted in the Python PYHELLOGUI.py module in the PYHELLOGUI sub-directory. The Makefile.am located in the src directory must be updated to include the PYHELLOGUI subdirectory. A Makefile.am must be added into the PYHELLOGUI subdirectory. Important targets are salomescript_SCRIPTS and salomeres_DATA.

The salomescript_SCRIPTS target must be updated with the name of the Python modules to be made visible in Salome, in other words mainly so that they are importable (Python import command).

The salomeres_DATA target must be updated with the names of files that are used for multi-linguism.

Menu bar and button bar

The menu bar and button bar for the PYHELLO component are dynamically added when importing the PYHELLOGUI module. They are created by calling the Python functions createMenu, createAction and createTool from the sgPyQt SALOME interface object. Every action must have a unique id. Some icons are used. They must be installed in the resources directory.