Archived:Embedding Python into C++ - Complex Description

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Introduction

Use of the Python scripting language is an excellent way how to extend the functionality of the C++ program. It is ideal especially for the game scripting or for complex configuration files. The Python itself can be downloaded from the *sourceforge.net* pages. For the embedding on the S60 3rd platform the sis file (`PythonForS60_1_4_5_3rdEd.sis`) is necessary to be installed on the phone. In fact the only necessary component for the embedding is the Python interpreter DLL library `python222.dll` included within the installation sis file. If embedding the sis file into application sis, which invokes the Python from C++, is not the suitable way or capabilities of the DLL must be changed or there is simply some other reason to not use the installation directly, we have the possibility to download sources (`pys60-1.4.5_src.zip`) and build the python interpreter DLL on our own. With the help of description in the `readme.txt` from the sources zip file it is quite easy. If the own build of the interpreter DLL will be part of the sis installation package you have to remember to change its UID and name to not clash with the original Python installation.

Portability

The advantage of the Python is also its availability on Windows Mobile platform, so if you are writing the portable C++ application it is currently the only choice (as far as I know) for the scripting language as other good languages like `Lua` or `GameMonkey` are not ported to both platforms.

Embedding

For embedding scripts into C++ code there are two possibilities, either use directly the Python/C API or use the `CSPyInterpreter` Symbian wrapper class. While use of the wrapper class is easier to use, it does not allow handling input/output script (module) parameters, calling selected functions and getting back the result. To be able to use the Python in C++ code the SDK from the www.sourceforge.net pages has to be downloaded and installed. For each S60 3rd platform edition there is the separate package (e.g. `PythonForS60_1_4_5_SDK_3rdEd.zip`) – it contains include header files and the `Python222.lib` export library we need to link against.

Using the `CSPyInterpreter` class is pretty simple, as shown in the code snippet below:

```
...
RunPythonSimpleScriptL( _L("c:\\resource\\writeToFile.py") );
...
void RunPythonSimpleScriptL( const TDesC& aScriptName )
{
  // Create a Python interpreter
  CSPyInterpreter* it = CSPyInterpreter::NewInterpreterL();
  CleanupStack::PushL( it );

  // Convert the script name to char*
  HBufC8* scriptName = CnvUtfConverter::ConvertFromUnicodeToUtf8L( aScriptName );
  CleanupStack::PushL( scriptName );
  char* scriptNameChar = (char*)scriptName->Des().PtrZ();

  // Run script
  TInt err = it->RunScript( 1, &scriptNameChar );
  User::LeaveIfError( err );

  // Clean everything
  CleanupStack::PopAndDestroy( scriptName );
  CleanupStack::PopAndDestroy( it );
}
```

Python script `writeToFile.py`:

```
file = open("c:\\test.txt", 'w')
file.write("This is the new content of test.txt :-)")
file.close()
```

In a more complex example we need:

- Convert data values from Symbian C++ to native C representation and then to Python representation
- Perform a function call from a Python script using converted values
- Convert data values from the Python back to the Symbian C++

Following code snippets do the stuff. First some initialization is done; the `c:\scripts` path is specified as a script repository. If the path is not specified `.py` scripts are expected to be saved in the `c:\resource` directory. Then all parameters are converted to the `PyObject` type and the script is loaded by calling the `PyImport_Import()` method. Later on the function object is retrieved from the script and is called. The result is converted from `PyObject` to `char*` and then to `TPtrC8`.

(C) Copyright Nokia 2013. All rights reserved.

```
void RunPythonScript1L()
{
  // Create a Python interpreter
  CSPyInterpreter* it = CSPyInterpreter::NewInterpreterL();
  CleanupStack::PushL( it );

  // Save state of any current Python interpreter, and acquire the
  // interpreter lock
  PyEval_RestoreThread( PYTHON_TLS->thread_state );

  // Set path for .py scripts
  _LIT8(KPyExecPath, "c:\\scripts\0");
  PySys_SetPath((char*)TPtrC8(KPyExecPath).Ptr());

  // .py module name
  char *module_name = "simpleScript";

  // .py module function name
  char *function_name = "returnString";


  // Python API objects
  PyObject *py_module_name, *py_module, *py_function, *py_result;

  // Module (script) name (Return a new string object with a copy of the string)
  py_module_name = PyString_FromString( module_name );

  // The function imports the module name, potentially using the given
  // globals and locals to determine how to interpret the name in a package context
  py_module = PyImport_Import( py_module_name );
  Py_DECREF( py_module_name );

  if ( py_module )
  {
    // Retrieve an attribute named from object (i.e. function from module)
    py_function = PyObject_GetAttrString( py_module, function_name );

    if ( py_function )
    {
      // Call a callable Python object
      py_result = PyObject_CallObject( py_function, NULL );
      Py_DECREF( py_module );
      Py_DECREF( py_function );

      // Convert result from PyObject to char*
      int result_len = -1;
      // .py function result
      char *result = NULL;

      result = PyString_AsString( py_result ) ;
      result_len = strlen( result ) ;

      // Make a Symbian descriptor pointer to the char * response
      TPtrC8 sym_result( (TUint8*)result, result_len );
      if ( sym_result.Compare( _L8("Hello world") ) != 0 )
      {
        // The result is other then expected
        Py_DECREF( py_result );
        PyEval_SaveThread();
        User::Leave( KErrGeneral );
      }

      Py_DECREF( py_result );
    }
    else
    {
      // Function not found in the module
      Py_DECREF( py_module );
      PyEval_SaveThread();
      User::Leave( KErrNotFound );
    }

  }
  else
  {
    // Module not found
    PyEval_SaveThread();
    User::Leave( KErrNotFound );
  }

  PyEval_SaveThread();
  CleanupStack::PopAndDestroy( it );
}
```

The second example shows the situation, where the method to sum two number is called, so we need to pass to Python script two integer values we want to be added. This is done by `PyTuple_XXX()` methods, which construct kind of dynamic array into which we consequently insert `PyObject` arguments for the script function.

```
void RunPythonScript2L()
{

  // Create a Python interpreter
  CSPyInterpreter* it = CSPyInterpreter::NewInterpreterL();
  CleanupStack::PushL( it );

  // Save state of any current Python interpreter, and acquire the
  // interpreter lock
  PyEval_RestoreThread( PYTHON_TLS->thread_state );

  // Set path for .py scripts
  _LIT8(KPyExecPath, "c:\\scripts\0");
  PySys_SetPath((char*)TPtrC8(KPyExecPath).Ptr());

  // .py module name
  char *module_name = "simpleScript";

  // .py module function name
  char *function_name = "sum";

  //  .py function result
  int result = -1;


  // Python API objects
  PyObject *py_module_name, *py_module, *py_function, *py_result;
  // Python funtcion agum,ents
```

 (C) Copyright Nokia 2013. All rights reserved.

```
  PyObject *py_arguments, *py_argument;

  // Module (script) name (Return a new string object with a copy of the string)
  py_module_name = PyString_FromString( module_name );

  // The function imports the module name, potentially using the given
  // globals and locals to determine how to interpret the name in a package context
  py_module = PyImport_Import( py_module_name );
  Py_DECREF( py_module_name );

  if ( py_module )
  {
    // Retrieve an attribute named from object (i.e. function from module)
    py_function = PyObject_GetAttrString( py_module, function_name );

    if ( py_function )
    {

      // Create fumction arguments object of size 2
      py_arguments = PyTuple_New( 2 );

      // 1st argument
      py_argument = PyInt_FromLong(5);
      PyTuple_SetItem(py_arguments, 0, py_argument);

      // 2nd argument
      py_argument = PyInt_FromLong(3);
      PyTuple_SetItem(py_arguments, 1, py_argument);

      // Call a callable Python object
      py_result = PyObject_CallObject( py_function, py_arguments );
      Py_DECREF( py_arguments );
      Py_DECREF( py_module );
      Py_DECREF( py_function );

      // Convert result from PyObject to char*
      result = PyInt_AsLong( py_result ) ;
      if ( result != 8 )
      {
        // The result is other then expected
        Py_DECREF( py_result );
        PyEval_SaveThread();
        User::Leave( KErrGeneral );
      }

      Py_DECREF( py_result );
    }
    else
    {
      // Function not found in the module
      Py_DECREF( py_module );
      PyEval_SaveThread();
      User::Leave( KErrNotFound );
    }

  }
  else
  {
    // Module not found
    PyEval_SaveThread();
    User::Leave( KErrNotFound );
  }

  PyEval_SaveThread();
  CleanupStack::PopAndDestroy( it );
}
```

The `simpleScript.py`:

```
# Returns the string simply
def returnString():
  return "Hello world"

# Returns sum of 2 given numbers
def sum(a,b):
    c = a + b
    return c
```

Links

Sourceforge (until version 1.4.5): http://sourceforge.net/projects/pys60/files/pys60/

Maemo Garage (for newer versions): https://garage.maemo.org/frs/?group_id=854&release_id=3264

Python/C API: http://docs.python.org/extending/embedding.html

The article can be found also here: http://stenlikmobile.blogspot.com/ Nokia Developer Wiki:Copyrights

 (C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.