Bringing a C library into the Future

Dirk Haun
Compart Systemhaus GmbH
ACCU Conference 2005

updated slides: http://www.haun-online.de/accu/

About Compart

Compart Systemhaus GmbH is a supplier of products for document and output management.

Typical application scenarios are document viewing, conversion, formatting, re-organizing document spools, classifying and indexing, changing page information, controlling and optimizing data streams in print spools.

Products are designed for large document volumes demanding high transaction rates.

Motivation

- Company knowledge (aka "IP") accumulated in existing code
- Tried and trusted code
- Originally implemented in one language

  (in our case: in C, for speed and portability)
- Need to use in other languages/ environments

So what can we do?

Option #1: Rewrite

- ☐ A lot of work
- ☐ Possible introduction of bugs while porting
- ☐ Multiple code bases to maintain
- ☐ Possible performance impacts
- ☑ Allows for best possible native interface

Option #2: Reuse

- ☑ Provide bindings
- ☑ Use of a code generator
- ☑ Overall faster to implement
- ☑ Development can continue on the original code - we only have to update the bindings
- ☐ Possibly not the best native interface

What we have

Sample of existing code*

```
PCPDOCPONENT pPage;

DcpPageCreate(&pPage);

DcpPageSetRotation(pPage, 90);

DcpPageDestroy(pPage);
```

* error handling left out

The concept

Using an IDL

- XML based Interface Definition Language
- Describe methods and their parameters
- Language-specific overrides / additions
- Documentation

IDL: Functions

```
<class name="DcpPage">

<function name="SetRotation" type="cprc"
          comment="Rotate page">
    <parameterlist>
        <this/>
        <parameter name="Rotation" type="int"
                   comment="Rotation, in degrees"/>
    </parameterlist>
</function>

</class>
```

IDL: C'tor & D'tor

```
<constructor>
    <parameterlist>
        <this/>
    </parameterlist>
</constructor>

<destructor>
    <parameterlist>
        <this/>
    </parameterlist>
</destructor>
```

IDL: Types

```
<extdeflist>
    <extdef key="length"   name="Length"     type="int"
            jtype="Length" ctype="PCPLENGTH"/>
</extdeflist>

...

<function ...>
    <parameterlist>
        <extref name="Height" type="length"/>
```

IDL: Constants

```
<constlist name="CPDOCPONENTTYPE" jname="Type">
    <constitem name="Queue"     type="int"  value="0"/>
    <constitem name="Document"  type="int"  value="1"/>
    <constitem name="Sheet"     type="int"  value="2"/>
    <constitem name="Page"      type="int"  value="3"/>
    <constitem name="Omr"       type="int"  value="4"/>
    <constitem name="Barcode"   type="int"  value="5"/>
</constlist>
```

IDL: Language-specifcs

```
<snipplet pass=".jsi.c">

#include &lt;cpbase.h&gt;
#include &lt;cpdocpon.h&gt;
#include &lt;jsapi.h&gt;

#include "jscputil.h"

</snipplet>
```

IDL: Documentation

```
<doc pass=".jsi.c">
  <description>Length constructor</description>
  <param name="pszValue">
    <para>length string (value + unit)</para>
  </param>
  <return>new Length object</return>
  <example>length = new Length("2in");</example>
</doc>
```

IDL: Miscelleanous

- Primitive types are built in: int, boolean, string
- Parameters can be marked “optional”
- When the function takes ownership of an object, parameters have to be marked “keepref” to control garbage collection,

  e.g. Document.put(Page);

An IDL example

```
<?xml version="1.0" standalone="yes"?>
<cpidl>
<class name="DcpPage" jname="Page" baseclass="Docponent" package="compart.cplib.docponent">

    <thisdef name="DcpPage" ctype="PCPDOCPONENT" type="int"/>

    <constructor>
        <parameterlist>
            <this/>
        </parameterlist>
    </constructor>

    <destructor>
        <parameterlist>
            <this/>
        </parameterlist>
    </destructor>

    <function name="SetRotation" type="cprc" comment="Rotate page">
        <parameterlist>
            <this/>
            <parameter name="Rotation" type="int" comment="Rotation, in degrees"/>
        </parameterlist>
    </function>
</class>
</cpidl>
```

What we get

JavaScript example

```
page = new Page();

page.setRotation(90);

delete page;
// or garbage collection
```

C code for JavaScript

```
JSBool Page_SetRotation(JSContext *cx, JSObject *obj, uintN argc, jsval *argv, jsval *rval)
{
JSBool result = JS_FALSE;
CPRET CpRc = CP_OK;
CPJSPRIVATEDATA *private = NULL;
PCPDOCPONENT iDcpPage = NULL;
CPSIZE_T iRotation = 0;
CPINT iRotationArgv;

for(EVER) {
    if(argc != 1) {
        CPJS_ThrowException(cx, "Expected 1 parameter but got %d.", argc);
        result = JS_FALSE;
        break; }

    private = (CPJSPRIVATEDATA*) JS_GetPrivate(cx, obj);
    iDcpPage = (PCPDOCPONENT) (private->pSelf);

    CPJS_GetArgumentAsInteger(cx, argv[0], "Page.SetRotation", 0, &iRotationArgv);
    iRotation = (CPSIZE_T) iRotationArgv;

    CpRc = DcpPageSetRotation(iDcpPage, iRotation);
    if(CpRc != CP_OK) {
        CPJS_ThrowException(cx, "Call to underlying cplib function failed with CpRc=%d", CpRc);
        result = JS_FALSE;
        break; }

    result = JS_TRUE;
    break; }

return(result);
}
```

JNI code

```
JNIEXPORT void JNICALL  Java_compart_cplib_docponent_Page_j2cSetRotation
               (JNIEnv *prmpJNIEnv, jclass  prmjclass, jint jiDcpPage, jint jiRotation)
    {
     CPRET       CpRc;

    /* declare native C vars - conversion of jni/c formal parameters */
    PCPDOCPONENT    iDcpPage=(PCPDOCPONENT)0;
    CPSIZE_T    iRotation=(CPSIZE_T)0;

    UNUSED(prmjclass); // class object currently not used

    TrcEntry(TRCHOOK, ">Java_compart_cplib_docponent_Page_j2cSetRotation");

    /* convert jni formal parameters to C variables                        */
    iDcpPage    = (PCPDOCPONENT)jiDcpPage;
    iRotation   = (int)(jiRotation);
    /* invocation of native C function                                     */
    CpRc = DcpPageSetRotation(iDcpPage, iRotation);
    if(CpRc != CP_OK)
        {
        CPBOOL fException;
        /* check return code and conditionally throw exception                */
        /* go immediate to end on exeption (cleanup may dilluate exception stack) */
        fException = CpjJNICBindingThrowException(prmpJNIEnv, CpRc, "DcpPageSetRotation");
        if (fException) goto functionexit;
        }

    TrcExit(TRCHOOK, "<Java_compart_cplib_docponent_Page_j2cSetRotation");
    functionexit:

    return;
    }
```

Java example

```
    /**
     * @param Rotation
     *        Rotation, in degrees
     */
    public void setRotation
        (
        int Rotation
        )
     throws compart.cplib.cjava.JCPLib.Exception
        {
        j2cSetRotation(this.cjavaHandle, Rotation);
        }
```

Runtime reference

```
js> p = new Page()
[object Page]
js> explain(p);
Function replaceVars // Replace variables in stamp files
    1: Vars jihVars;

Function setRotation // Rotate page
    1: int jiRotation; // Rotation, in degrees
```

Supported languages

- Java (JNI + Java code)
- JavaScript
- Ruby
- C
- Definition files
- DocBook

Summary

- Reuse of existing code - unchanged
- Provides bindings for other languages
- Generated automatically
- Supports language-specific concepts

Looking back ...

- A lot of effort went into getting the language concepts right
- At least some of the interface definitions could be created with XSLT scripts

Looking ahead ...

- Support for other languages / environments, e.g. .NET
- More control over the code generation
- Cleanup of minor quirks
- More generic code generation: Describe datastructures and create getter / setter methods

The End