

Work with InterSystems.

Not separate systems.



Code Generation with Cache' Object Script

Alain Houf Sales Engineer - InterSystems

INTERSYSTEMS

Code Generation with Cache' Object Script



- Class Generation
- Method Generation
- Routine Generation

Work with InterSystems.

Not separate systems.



Class Generation

What is a Class Generator?



- A program that
 - reads a external metadata source and generates xml files that contain class definitions.
 - Imports the files and generates the business logic.



- Abstract class **%SYSTEM.SQL**
Classmethod **DDLImport(DDLMode As %Library.String,.....)**
Parameters: *DDLMode*

Vendor from which the script file originated. This parameter is required. Supported values are:

FDBMS
Informix
Interbase
MSSQLServer
Oracle
Sybase

Class Generator



- Not only for Table definitions but also for stored procedures
 - Support for TSQL in Caché
 - Stored procedure convertor for Informix in Caché 2008/2009



Custom written Class Generators

Creates xml-files with following syntax

```
{
  set xmlversion="<?xml version="_"_""1.0""_<< encoding="_"_""UTF-
8""_"_?>"

  Do file.WriteLine("<Classname="_"_""_
$translate(tablename,"_"_,"")_"_"">")
  Do file.WriteLine("<ClassType>persistent</ClassType>")
  ...

  Do file.WriteLine("</Export>")
  Do file.Close()
}
```

Class Generator



Load and compile the file and check for syntax errors

```
set status = $system.OBJ.Load(xmlfile,"crkf-d")
if 'status {
  do ##class(User.Import).AddError("Problem compiling
xml-file: "_xmlfile)
}
```

Work with InterSystems.

Not separate systems.



Method Generation

INTERSYSTEMS

What is a Method Generator?



- A small program that is invoked by the class compiler to generate the runtime code for a method.
- A method generator is simply a method of a Caché class that has its CodeMode keyword set to "objectgenerator" .
Generated Code

A simple method generator



```
Class MyApp.MyClass Extends %RegisteredObject
{
Method MyMethod() [ CodeMode = objectgenerator ]
{
    Do %code.WriteLine(" Write "" _ %class.Name _ """)
    Do %code.WriteLine(" Quit")
    Quit $$$OK
}
}
```

Output from the simple generator



- When the class MyApp.MyClass is compiled

```
Method MyMethod()
```

```
{  
    Write "MyApp.MyClass"  
    Quit  
}
```

How Method Generators Work



- Resolves inheritance for the class
 - builds a list of all inherited members.
- Makes a list of all methods specified as method generators
 - by looking at the CodeMode keyword of each method.
- Gathers the code from all method generators, copies it into one or more temporary routines, and compiles them
 - makes it possible to execute the method generator code.
- Creates a set of transient objects that represent the definition of the class being compiled.
 - These objects are made available to the method generator code.
- It executes the code for every method generator.

How Method Generators Work



- If present, the compiler will arrange the order in which it invokes the method generators by looking at the value of the GenerateAfter keyword for each of the methods.
 - This keyword gives you some control in cases where there may be compiler timing dependencies among methods.
- Copies the results of each method generator (lines of code plus any changes to other method keywords) into the compiled class structure (used to generate the actual code for the class).
- the original method signature (arguments and return type), as well as any method keyword values, are used for the generated method.
 - If you specify a method generator as having a return type of %Integer, then the actual method will have a return type of %Integer.
- Generates the executable code for the class by combining the code generated by the method generators along with the code from all the non-method generator methods.

Method Generators - Context



- %code
 - %Stream.MethodGenerator class. This is a stream into which you write the code for method.
- %class
 - %Dictionary.ClassDefinition class. It contains the original definition of the class being compiled.
- %method
 - %Dictionary.MethodDefinition class. It contains the original definition of the method being compiled.
- %compiledclass
 - %Dictionary.CompiledClass class. It contains the compiled definition of the class being compiled. It contains information about the class after inheritance has been resolved (such as the list of all properties and methods, including those inherited from superclasses).
- %compiledmethod
 - %Dictionary.CompiledMethod class. It contains the compiled definition of the method being generated.



So how does this work in practice?

TM.MyMethods



```
ClassMethod ListProperties() [ CodeMode = objectgenerator ]
{
For i = 1:1:%compiledclass.Properties.Count()
    {
        Set prop = %compiledclass.Properties.GetAt(i).Name
        Do %code.WriteLine(" Write "" _ prop _ "" ,!")
    }
Do %code.WriteLine(" Quit")
Quit $$$OK
}
```



So how does this work in practice? - .MAC



```
#classmethod ListProperties
ListProperties(%class,%code,%method,%compiledclass,%compile
dmethod,%parameter) public {

    For i = 1:1:%compiledclass.Properties.Count()
    {
        Set prop = %compiledclass.Properties.GetAt(i).Name
        Do %code.WriteLine(" Write "" _ prop _ "" ,!")
    }
    Do %code.WriteLine(" Quit")
    Quit $$$OK
q 1 }
```

NewClass (%Persistent, TM.MyMethods)



- When the class NewClass is compiled, a ListProperties method is generated. You can see this in the NewClass1.int

```
zListProperties() public { Write "%Concurrency",!  
Write "Name",!  
Write "SSN",!  
Quit }
```



- Call
 - this method is an alias for a routine call (used for wrapping legacy code).
- Code
 - this method is implemented as lines of code (the default).
- Expression
 - this method is implemented as an expression.
- Objectgenerator
 - this method is a method generator.

CodeMode - expression



```
/// Converts <var>%val</var>, which represents a logical time value  
/// (number of seconds since midnight), into ODBC time format.  
/// <p>Returns the ODBC time string for the logical time value <var>%val</var>.
```

```
ClassMethod LogicalToOdbc(%val As %Time = "") As %String [  
CodeMode = expression, ServerOnly = 1 ]
```

```
{
```

```
$select(%val="":",1:$ztime(%val))
```

```
}
```

A simple property with %Time



```
Class TM.NewClass1 Extends %Persistent
{
Property NewProperty1 As %Time;
}
```

The compiled property has a generated method

```
zNewProperty1LogicalToOdbc(%val="") Quit select(%val="": "", 1:$ztime(%val))
```

Implementing Parameters with Code Generation -TechStatus



```
/// Active Status
```

```
Property TechStatus As SQ.MHMultiChoice(DISPLAYLIST =  
",Active,Inactive,Active", MAXLEN = 8, SELECTIVITY =  
"25.0000%", VALUELIST = ",,N,Y") [ SqlColumnNumber = 3 ];
```

LogicalToOdbc method - MultChoice



```
ClassMethod LogicalToOdbc(%val As %String) As %String [ CodeMode = generator ]  
{
```

```
  n i,len,sep  
  s %code=0
```

```
  i %parameter("VALUELIST")!="" ,%parameter("DISPLAYLIST")!="" d QUIT $$$OK  
  . s sep=$e(%parameter("VALUELIST"))  
  . s len=$l(%parameter("VALUELIST"),sep)  
  . f i=2:1:$l(%parameter("VALUELIST"),sep) $$$GENERATE("  
q:%val="_$$quote($p(%parameter("VALUELIST"),sep,i))_"_"$$quote($p(%parameter  
  . $$$GENERATE(" q """""))
```

```
  s %codemode=$$$cMETHCODEMODEEXPRESSION  
  s %code="%val"  
  QUIT $$$OK
```

```
}
```

Implementing Parameters with Code Generation -TechStatus



```
/// Active Status  
Property TechStatus As SQ.MHMultiChoice(DISPLAYLIST =  
",Active,Inactive,Active", MAXLEN = 8, SELECTIVITY = "25.0000%", VALUELIST =  
",,N,Y") [ SqlColumnNumber = 3 ];
```

```
zTechStatusLogicalToOdbc(%val) q:%val="" "Active"  
q:%val="N" "Inactive"  
q:%val="Y" "Active"  
q ""
```

Work with InterSystems.

Not separate systems.



Routine Generation

INTERSYSTEMS

What is a Routine Generator?



- A program that generates at runtime (creates/compiles) code to be executed.
- This routines can contain COS, embedded SQL and Object methods.

Routine Generator



```
Set routine =##class(%Routine).%New("MyRoutine.MAC")
    //Write lines of code to the routine
Do routine.WriteLine("MyRoutine")
Do routine.WriteLine("Tag()")
Do routine.WriteLine(" Write ""This is my routine",!)")
Do routine.WriteLine(" Quit") ;
    //save the routine
Do routine.Save() ;
    //compile the routine
Do routine.Compile() ;
    // close the routine object
Do routine.%Close()
```



Make use of the %Dictionary Classes

- `##class(%Dictionary.CompiledClass).%OpenId(classname)`
 - `Properties.GetAt(x)` (`%Dictionary.CompiledProperty`)
 - `Storages.GetAt(x)` (`%Dictionary.CompiledStorage`)
 - `Indices.GetAt(x)` (`%Dictionary.CompiledIndex`)

Routine Generator



Use **\$ZOBJMETHOD**, **\$ZOBJCLASSMETHOD** and **\$ZOBJPROPERTY** for more generic way to refer to class logic and properties

```
set ClassmethodName = "%OpenId"  
set obj=$ZOBJCLASSMETHOD(tabelnaam, ClassmethodName, Key)  
set objp=$ZOBJPROPERTY(obj,propnaam)  
if (objp) {  
    set id=$ZOBJMETHOD(objp, "%Id")  
    set classname=$ZOBJMETHOD(objp, "%ClassName",2)  
}
```



- Examples of method generation
 - %Library.Persistent, %Library.PopulatUtils, %XML.Adapter
- Examples of class generation
 - [ftp.intersystems.com](ftp://intersystems.com)
 - /pub/dbms/FDBMSToCDL_41_009.zip
 - /pub/dbms/FileMan2Cache_5_0_10.zip
- Method Generator documentation
 - <http://www.intersystems.com/cache/downloads/documentation.html>
 - Keyword - method generator

Examples



- Generation of Methods
 - User.email as datatype that extends %String
 - Abstract class with methods
 - %OpenPk
 - %ExistsPk
- Used @ Spector, Fed Pol, Iknow

Examples



- Generation of ClassDefinitions
 - Complete classdefinition exists in Excel -file
 - Generation of XML files from this file
 - Loading, compiling and tune of the tables
- Used @ Fed Pol for datawarehouse and questis servers

Examples



- Generation of Routines
 - Class User.Import, methods
 - GeneratieRoutine
 - GeneratieRoutineMulti
- Used @ Fed Pol, replacement of +270 Stored procedures by 1 single routine that generates and executes the load logic

Work with InterSystems.

Not separate systems.



Thank You!!

INTERSYSTEMS