InterSystems Caché

Adding Compiled Code to Customer Databases

Version 2018.1
2024-04-03

InterSystems Corporation   1 Memorial Drive   Cambridge MA 02142   www.intersystems.com

Adding Compiled Code to Customer Databases
Caché Version 2018.1 2024-04-03
Copyright © 2024 InterSystems Corporation
All rights reserved.

Table of Contents

Adding Compiled Code to Customer Databases

This article describes how to add compiled code to the customers' databases, so that you can provide your customers with new code that does not need to be recompiled. It discusses the following topics:

- Requirements
- How to deploy compiled code
- Limitations
- Example
- Effect on running processes

Important: InterSystems highly recommends that you test this procedure with your specific classes in a test environment before using it on a live system.

1 Requirements

The requirements are as follows:

- The Caché version must be the same on the system where you create and compile the code as it is on the system where you are installing it.
- The SQL delimited identifier setting must be the same on both systems.

2 Deploying Compiled Code

To deploy compiled code, do the following:

1. Create a Studio project that contains class definitions, routines, and globals.

   You can create the project in Studio or programmatically with methods of the %Studio.Project class.
2. Make sure that all the code in the project has been compiled.
3. Obtain an OREF to the project, by using methods of %Studio.Project.
4. Call the **DeployToFile()** instance method of the project. For example:

   ObjectScript

   ```
    set sc=projectoref.DeployToFile("c:\test\myglobal.xml",,1)
   ```

Because the third argument is 1, the generated file does not contain the source code and intermediate code. The following shows the method signature and details:

```
method DeployToFile(file As %String,
                   qspec As %String,
                   removesource As %Boolean = 0) as %Status
```

Generates a file that contains the packaged code belonging to this project. *file* is the name of the file, and *qspec* is a string containing the standard compile qualifiers; see "Flags and Qualifiers" in the reference for $SYSTEM. Note in particular that you should ensure that the `/exportselectivity` qualifier is specified as you want it. You can also specify the `k` flag, which controls whether the source code of generated routines is kept.

removesource is a boolean value. If *removesource* is 1, the global does not contain the routine and method sources, nor the intermediate code (regardless of the setting of the `k` flag).

For information on deployed mode, see "Making Classes Deployed" in *Using Caché Objects*.

5. Provide this file to your customers with instructions. Customers should use the Terminal, switch to the appropriate namespace, and call the **InstallFromFile()** method of %Studio.Project, as follows:

   ```
   set sc=##class(%Studio.Project).InstallFromFile("c:\test\myglobal.xml")
   ```

As of release 2017.1, **DeployToFile()** automatically adds, to the list of items being deployed, any parent classes and any child classes referenced by relationships in this class.

3 Limitations for DeployToFile() and InstallFromFile()

The **DeployToFile()**and **InstallFromFile()** methods have the following limitations:

- The **DeployToFile()** and **InstallFromFile()** methods do not handle any class projections, for example, classes that project to Java files.
- These methods do not do any dependency checking and thus do not automatically include subclasses. It is your responsibility to insert all the classes you need into the package.

  For example, **InstallFromFile()** does not check the location into which it installs the code. As an additional example, you could potentially deploy a subclass to a system that does not have the superclass installed.

Important: InterSystems highly recommends that you test this procedure with your specific classes in a test environment before using it on a live system.

4 Example

The following example shows a simple routine that exports the code for the Sample.Customer and Sample.Person classes; you can use this in the SAMPLES namespace. Note that this routine creates a project but does not save it; there is no need to save the project before calling **DeployToFile()**.

```
 ; deployexample
 set p=##class(%Studio.Project).%New()
 do p.AddItem("Sample.Customer.cls")
 do p.AddItem("Sample.Person.cls")

 do p.DeployToFile("c:\test\myglobal.xml",,1)
```

For demonstration purposes, this simple routine does not include error checking.

5 Effect on Running Processes

This section describes the effect on any running processes when you load a new, compiled version of the code into a system where code is currently executing:

- Routines and class methods that are currently executing will continue to use the old code. If the routine or class method is called again from the same process after the compilation is complete, it will load the new code.
- If a routine or class method invokes another routine or class method and then returns to the caller, the process continues to use the old version of the calling code. For example, suppose that routine A invokes routine B, after which control returns to routine A. If you recompile routine A while routine A is running (or while routine B is running), when control returns to routine A, the process continues to use the old code for routine A.
- Any open process continues to use the in-memory version of any OREFs.
- Instance methods are dependent on an instance of an object. Any object will use the version of the instance methods that existed when the object was instantiated. The object will continue to use that code until the OREF is discarded and a new object is instantiated. So if you instantiate an object and then import a new version of the class (or delete it totally), you can still continue to call methods on the instance, which will use the version of the class that existed when the object was instantiated.

Note that you can query an OREF to see if the instance is using the latest version of the class code:

ObjectScript

```
 Write oref.%ClassIsLatestVersion()
```

The **%ClassIsLatestVersion()** method returns 1 if the instance is using the latest version of the class code; otherwise it returns 0.

The preceding comments also apply when you recompile code in a given instance.

Note: The comments here apply on the instance where you are compiling the class. On another instance over ECP or an mirror member, the routine or class can briefly be inconsistent and can throw errors.