Using the Caché %Installer Class

Version 2009.1
30 June 2009

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

Using the Caché %Installer Class
Caché Version 2009.1 30 June 2009
Copyright © 2009 InterSystems Corporation
All rights reserved.

Table of Contents

Using the Caché %Installer Class

The %Installer class, which is available in %SYS namespace, provides the ability to create a class that defines an installation manifest which can be executed via a class method. %Installer xml tags are used to define an installation manifest that generates the code to execute the manifest instructions.

This article provides information about defining namespaces, databases, and mappings (global, routine and package). Classes, routines and globals can be loaded into a namespace defined in the installation manifest. Then %Installer can compile the classes and routines making them available for execution in the namespace in which they are loaded.

Note: For an example of a user-defined installer class, see Sample.Installer in the SAMPLES namespace.

1 Using the %Installer Class

The overall structure of an %Installer class is:

```
Include %occInclude
Class MyPackage.MyInstaller
{
XData MyInstall [ XMLNamespace = INSTALLER ]
   {
   <Manifest>
      <Var/>
      <If/>
      <Namespace>
         <Configuration>
            <Database/>
            <GlobalMapping/>
            <RoutineMapping/>
            <ClassMapping/>
         </Configuration>
         <If/>
         <Import/>
         <Invoke/>
      </Namespace>
   </Manifest>
   }
}
ClassMethod setup(ByRef pVars, pLogLevel As %Integer, pInstaller As
   %Installer.Installer) As %Status [ CodeMode = objectgenerator,
   Internal, Private ]
   {
   #; Let our XGL document generate code for this method.
   Quit ##class(%Installer.Manifest).%Generate(%compiledclass,
      %code, "MyInstall")
   }
```

The %occInclude include file, which includes methods required by the %Installer class, must be specified at the beginning of every %Installer class you create.

The name of the XData block (MyInstall) is used by the **MyPackage.MyInstaller.setup** method to generate the code for that manifest, which is passed as the third argument of the **%Installer.Manifest.%Generate** method.

Note: Referencing `XMLNamespace = INSTALLER` provides Studio Assist capability while authoring your %Installer class.

The outer-most xml tag, `<Manifest>`, contains all the information for code generation. There can be as many `<Namespace>` tags as needed within the `<Manifest>` tag to define namespaces in the manifest, but there must be at least one.

There must be one `<Configuration>` tag within each `<Namespace>`tag, and as many `<Database>` tags within the `<Configuration>` tag as needed to define the namespace. In addition, following each `<Database>` tag, you can add `<GlobalMapping>`, `<RoutineMapping>`, and `<ClassMapping>` tags as required.

Within the context of a `<Namespace>` tag — after the databases and their mappings have been defined — globals can be loaded, and routines and classes can be loaded and compiled using `<Import>` tag. Class methods can be invoked using the `<Invoke>` tag. Invoked class methods can execute routines and access globals that have been imported.

Optionally, you can define variables with the `<Var>` tag; each variable must specify a name and value. When the value for the `<Var>` is needed, the name is referenced by the `${NameAssigned}` syntax.

2 %Installer Tags

The following table describes the tags you can use in your %Installer class.

%Installer Tags

Tag	Description

`<Manifest>`	Required. Contains the installation instructions (all other tags). For example:
	`<Manifest>`
	`<Namespace ... >`
	`<Configuration>`
	`</Configuration>`
	`</Namespace>`
	`</Manifest>`

`<Var>`	Optional; within `<Manifest>`. Defines variables that can be used with the manifest. For example:
	`<Var Name="MYAPPDIR" Value="${MYAPPDIR}/myapp"/>`
	`<Var Name="DBRESOURCE" Value=%DB_${NAMESPACE}"/>`
	`<Var Name="MYAPPRESOURCE" Value=%DB_MYAPP"/>`

`<If>`	Optional; within `<Manifest>` or `<Namespace>`. Defines conditional. For example:
	`<If Condition='$L("${NAMESPACE}")=0'>`
	`<Error Status="$$$NamespaceDoesNotExist">`
	`<Arg Value="${NAMESPACE}"/>`
	`</Error>`
	`</If>`

Tag	Description

`<Namespace>`	Required; within `<Manifest>`. One for each namespace. For example:
	`<Namespace Name="${NAMESPACE}"`
	`Create="yes"`
	`Code="${NAMESPACE}"`
	`Data="${NAMESPACE}"`
	`>`
	`<Configuration>`
	`<Database . . . />`
	`</Configuration>`
	`</Namespace>`

`<Configuration>`	Required; within `<Namespace>`. Contains namespace configuration tags for databases and mapping. For example:
	`<Namespace . . .>`
	`<Configuration>`
	`<Database . . . />`
	`</Configuration>`
	`</Namespace>`

Tag	Description

`<Database>`	Required; within `<Configuration>`. Defines one or more databases used in the namespace. For example:
	`<Namespace . . .>`
	`<Configuration>`
	`<Database Name="${NAMESPACE}"`
	`Dir="${MGRDIR}/${NAMESPACE}"`
	`Create="yes"`
	`Resource="${DBRESOURCE}"`
	`PublicPermissions=""/>`
	`<Database Name="MYAPP"`
	`Dir="${MYAPPDIR}/db"`
	`Create="no"`
	`Resource="${MYAPPRESOURCE}"`
	`PublicPermissions=""/>`
	`</Configuration>`
	`</Namespace>`

`<GlobalMapping>`	Optional; within `<Configuration>`. Defines global mapping for the namespace. For example:
	`<Namespace . . .>`
	`<Configuration>`
	`<Database . . . />`
	`<Database Name="MYAPP" . . . />`
	`<GlobalMapping Global="MyAppData.*" FROM="MYAPP"/>`
	`<GlobalMapping Global="cspRule" FROM="MYAPP"/>`
	`</Configuration>`
	`</Namespace>`

Tag	Description

`<RoutineMapping>`	Optional; within `<Configuration>`. Defines routine mapping for the namespace. For example:
	`<Namespace . . .>`
	`<Configuration>`
	`<Database . . . />`
	`<Database Name="MYAPP" . . . />`
	`<RoutineMapping Routines="MYAPP" Type="INC" FROM="MYAPP"/>`
	`</Configuration>`
	`</Namespace>`

`<ClassMapping>`	Optional; within `<Configuration>`. Defines package mapping for the namespace. For example:
	`<Namespace . . .>`
	`<Configuration>`
	`<Database . . . />`
	`<Database Name="MYAPP" . . . />`
	`<ClassMapping Package="MYAPP" FROM="MYAPP"/>`
	`</Configuration>`
	`</Namespace>`

`<Import>`	Optional; within `<Namespace>`. Defines globals, routines, and packages to be loaded (and routines and packages to be compiled). For example:
	`<Namespace . . .>`
	`<Configuration> . . . </Configuration>`
	`<Import File="${MYAPPDIR}/data/Defaults.gof"/>`
	`</Namespace>`

Tag	Description

`<Invoke>`	Optional; within `<Namespace>`. Defines class methods to execute routines and access globals that have been imported. For example:
	`<Namespace . . .>`
	`<Configuration> . . . </Configuration>`
	`<Invoke Class="Sample.Installer" Method="SetupDefaults" CheckStatus="1">`
	`<Arg Value="${NAMESPACE}"/>`
	`<Arg Value="${ISUPGRADE}"/>`
	`</Invoke>`
	`</Namespace>`

For a complete example of a user-defined installer class, see Sample.Installer in the SAMPLES namespace.

3 Invoking the %Installer Class Methods

You can invoke the %Installer class by doing either of the following:

- In the %SYS namespace, enter the following command in the Caché Terminal:

  ```
  Do ##class(<classname>).setup(<arglist>)
  ```

  where <*classname*> is the name of the %Installer class, and <*arglist*> specifies comma-separated named VARIABLE=<*value*> pairs, which are passed into the method and referenced by ${VARIABLE}. For example:

  ```
  %SYS> Do ##class(MyPackage.MyInstaller).setup()
  ```

- Export the %Installer class as DefaultInstallerClass.xml to the same directory where the Caché install (either .msi or cinstall) is run. It is loaded and compiled, and the **setup** method is executed.

  Note: If you use the export technique, you cannot pass arguments to the **setup** method.