Article
Eduard Lebedyuk · Feb 19, 2016 12m read

Deploying Applications in InterSystems Cache with %Installer

Suppose you have developed your own app with InterSystems technologies stack and now want to perform multiple deployments on the customers' side. During the development process you've composed a detailed installation guide for your application, because you need to not only import classes, but also fine-tune the environment according to your needs.
To address this specific task, InterSystems has created a special tool called %Installer. Read on to find out how to use it.

%Installer

Using this tool, you can define the installation manifest, which describes the desired Caché configuration instead of the installation steps. You need only to describe what you want, and Caché will automatically generate the necessary code to modify the environment for you. Therefore, you should distribute only the manifest itself, while all the installation code will be generated for the specific Caché server at compile time.

To define a manifest, create a new XData block with detailed description of the target configuration and then implement a method to generate Caché ObjectScript code for this XData block (this code is always the same). Once the manifest is ready, you can access it from a terminal or Caché ObjectScript code or automatically during Caché installation. The manifest must be executed in the %SYS namespace. Manifests can handle both system parameters (superport, OS, mgr directory, etc.) and arbitrary user-supplied parameters. In general, each installation class must meet the following requirements:

- Contain a link to %occInclude.inc
- Contain an XData block with the Caché server configuration
- The block can be have any valid name
- Add [XMLNamespace = INSTALLER] after the block's name if you want to see prompts from Studio
- The root item (there must only be one) is called <Manifest> and comprises all other items
- You also need to implement the setup() method which will generate the necessary program code for the XData block.

Installer basics

You can execute an installation manifest in several ways:

- In the %SYS namespace using a terminal or from a Caché ObjectScript code

```
do ##class(MyPackage.MyInstaller).setup()
```

- Automatically during the installation of Caché. To do this, export the installer's class into DefaultInstallerClass.xml stored in the folder with Caché installation package (i.e. where setupcache.exe or cinstall are stored). During Caché installation, this class will be imported into the %SYS namespace and executed via the setup() method.

Example

Let's consider a simple example. Create the class called **App.Installer** containing an installer that will generate a new namespace with the user-defined name, create default web app and import code into this new namespace:

```
Include %occInclude
Class App.Installer {

/// You can see generated method in zsetup+1^App.Installer.1
XData Install [ XMLNamespace = INSTALLER ]
{
<Manifest>
    <If Condition='(##class(Config.Namespaces).Exists("${Namespace}")=0)'>
        <Log Text="Creating namespace ${Namespace}" Level="0"/>
        <Namespace Name="${Namespace}" Create="yes" Code="${Namespace}" Ensemble="0"
Data="${Namespace}">
            <Configuration>
                <Database Name="${Namespace}" Dir="${MGRDIR}${Namespace}" Create="yes
"/>
            </Configuration>
        </Namespace>
        <Log Text="End Creating namespace ${Namespace}" Level="0"/>
    </If>

    <Role Name="AppRole" Description="Role to access and use the App" Resources="%DB_
CACHESYS:RW,%Admin_Secure:U" />

    <Namespace Name="${Namespace}" Create="no">
        <CSPApplication Url="/csp/${Namespace}" Directory="${CSPDIR}${Namespace}" Aut
henticationMethods="64" IsNamespaceDefault="true" Grant="AppRole" />
        <IfDef Var="SourceDir">
            <Log Text="SourceDir defined - offline install from ${SourceDir}" Level="
0"/>
            <Import File="${SourceDir}"/>
        </IfDef>
    </Namespace>
</Manifest>
}

///Entry point method, you need to call
/// At class compile time it generate Caché ObjectScript code from the manifest
/// After that you can run this installer from a terminal:
/// Set pVars("Namespace")="NewNamespace"
/// Set pVars("SourceDir")="C:\temp\distr\"
/// Do ##class(App.Installer).setup(.pVars)
ClassMethod setup(ByRef pVars, pLogLevel As %Integer = 0, pInstaller As %Installer.In
staller) As %Status [ CodeMode = objectgenerator, Internal ]
{
     Quit ##class(%Installer.Manifest).%Generate(%compiledclass, %code, "Install")
}
}
```

In this example, installer performs the following actions:

- Checks if a namespace with the same name as the value of Namespace variable exists (for clarity let's specify that the Namespace variable is set to NewNamespace)
- If not, then logs that a new namespace called NewNamespace will be created
- Defines a new namespace:
    - Name is NewNamespace
    - Creates a new namespace
    - Routines database is NewNamespace
    - Do not enable Ensemble
    - Globals database is NewNamespace

- Creates a new database
  - Name is NewNamespace;
  - Creates it in the mgr/NewNamespace folder (note that the MGRDIR variable is available by default)
- Creation of the namespace is completed and logged
- Creates new role: AppRole (with resources %DBCACHESYS:RW and %AdminSecure:U)
- Default web application /csp/NewNamespace is created (it also grants AppRole automatically)
- If SourceDir variable is defined then imports all files from there into NewNamespace

To run this installer in a terminal, execute the following commands:

```
Set pVars("Namespace")="NewNamespace"
Set pVars("SourceDir")="C:\temp\distr\"
Do ##class(App.Installer).setup(.pVars)
```

During execution terminal displays relevant information:

```
2016-02-17 19:26:17 0 App.Installer: Installation starting at 2016-02-17 19:26:17, Lo
gLevel=0
2016-02-17 19:26:17 0 : Creating namespace NewNamespace
2016-02-17 19:26:17 0 : End Creating namespace NewNamespace
2016-02-17 19:26:17 0 : SourceDir defined - offline install from C:\temp\distr\
2016-02-17 19:26:18 0 App.Installer: Installation succeeded at 2016-02-17 19:26:18
2016-02-17 19:26:18 0 %Installer: Elapsed time .545148s
```

To receive even more information about what is going on specify LogLevel (from 0 (default) to 3 (raw); higher number = more information).

```
Do ##class(App.Installer).setup(.pVars, 3)
```

Now let's talk about what can be done in installer manifest.

Availible nodes

A manifest is composed of the following items:

Node	Parent node	Attributes (default values)	Description
Arg	Invoke, Error	Value – value of an argument	Passes an argument into a method called via Invoke or Error
ClassMapping	Configuration	Package – a package to be mapped From – name of the source database used for mapping	Creates a class mapping from a database to the namespace which contains this Configuration item
Compile	Namespace	Class – names of classes for compilation Flags – compilation flags (ck) IgnoreErrors – ignore errors (0)	Compilers classes. Calls $System.OBJ.Compile(Class, Flags)
Configuration	Namespace		Used for creating namespaces and

			databases. Closing tag activates mappings and updates the CPF file
CopyClass	Namespace	Src — source class Target — target class Replace — remove the source class (0)	Copies or moves the source class definition to the target one
CopyDir	Manifest	Src — source directory Target — target directory IgnoreErrors — ignore errors (0)	Copies a directory
CopyFile	Manifest	Src — source file Target — target file IgnoreErrors — ignore errors (0)	Copies a file
Credential	Production	Name – name of the access credentials Username – user name Password – user password Overwrite – overwrite if the account already exists	Creates or overrides the access credentials
CSPApplication	Namespace	AuthenticationMethods – enabled authentication methods AutoCompile – automatic compilation (in CSP settings) CSPZENEnabled – the CSP/ZEN flag ChangePasswordPage – path to change password page CookiePath – session cookie path CustomErrorPage – path to custom error page DefaultSuperclass – default superclass DefaultTimeout – session timeout Description – description Directory – path to CSP files EventClass – event class name Grant – list of roles assigned upon logging into the system GroupById – group by Id property InboundWebServicesEnabled – inbound web services flag	Creates or modifies a web application. For details, refer to documentation and the Security.Applications class

		IsNamespaceDefault – Namespace Default Application flag LockCSPName – Lock CSP Name flag LoginClass – path to login page PackageName – package name property PermittedClasses – permitted classes propert Recurse – recurce flag (serve subdirectories) (0) Resource – resource required to access web app ServeFiles – service files property ServeFilesTimeout – time, in seconds, of how long to cache static files. TwoFactorEnabled – two-step authentication Url – name of the web application UseSessionCookie – use cookies for the session	
Database	Configuration	BlockSize – block size in bytes of the database ClusterMountMode – mount the database as a part of the cluster Collation – sorting order Create – whether to create a new database (yes/no/overwrite (yes)) Dir – directory Encrypted – encrypt database EncryptionKeyID – ID of the encryption key ExpansionSize – size in MB to expand by InitialSize – initial size MaximumSize – maximum size MountAtStartup – mount upon launch MountRequired – specifies that the database MUST be successfully mounted at startup. Name – database name PublicPermissions – public permissions Resource – resource StreamLocation – directory where the streams associated with this	Creates or modifies a database. For details, refer to documentation, the Config.Databases and SYS.Database classes

		database go.	
Default	Manifest	Name – variable name Value – variable value Dir – variable value (if this is a path to a folder/file)	Sets the variable value (if it is not set yet)
Else	Manifest, Namespace	Will be executed when the if statement is false	
Error	Manifest	Status – error code Source – source of the error	Throws an exception. Note that ${} and #{} syntax is not available
ForEach	Manifest	Index – variable name Values – list of values for the variable	Collection-controlled loop
GlobalMapping	Configuration	Global – global name From – name of the database for mapping Collation – sorting order (Caché Standard)	Maps a global
If	Manifest, Namespace	Condition – a conditional statement	Conditional if statement
IfDef	Manifest, Namespace	Var – variable name	Conditional if statement used when the variable is already set
IfNotDef	Manifest, Namespace	Var – variable name	Conditional if statement used when the variable is not set yet
Import	Namespace	File – file/folder for import Flags — compilation flags (ck) IgnoreErrors — ignore errors (0) Recurse – import recursively (0)	Imports files. Calls: $System.OBJ.ImportDir(File,,Flags,,Recurse) and $System.OBJ.Load(File, Flags)
Invoke	Namespace	Class – class name Method – method name CheckStatus – check the returned status Return – write the result into a variable	Makes a call to a method of a class with various arguments and returns the execution results
LoadPage	Namespace	Name – path to the CSP page Dir – folder with CSP pages Flags — compilation flags (ck)	Loads CSP file using $System.CSP.LoadPage(Name, Flags) and $System.CSP.LoadPageDir(Dir, Flags)

		IgnoreErrors — ignore errors (0)	
Log	Manifest	Level – logging level from 0 (minimum) up to 3 (detailed) Text – string up to 32,000 characters in length	Adds a message to a log when the logging level is greater or equal to the "level" attribute
Manifest			Root item. The only one root item in a manifest; contains all other items
Namespace	Manifest	Name – name of the namespace Create – whether to create a new namespace (yes/no/overwrite (yes)) Code – database for program code Data – database Ensemble – enable Ensemble for the namespace All other attributes are applicable to the Ensemble web applications	Defines the installer's scope
Production	Namespace	Name – production name AutoStart – automatic launch of the production	Configures the Ensemble production
Resource	Manifest	Name – resource name Description – resource description Permission – public permissions	Creates or modifies a resource.
Role	Manifest	Name – name of the role Description – role description (can't contain commas) Resources – resourcesgiven to role, represented as "MyResource:RW,MyResource1:RWU" RolesGranted – whether to grant the corresponding roles	Creates a new role
RoutineMapping	Configuration	Routines – routine name Type – routines type (MAC, INT, INC, OBJ or ALL) From – source database	Creates a new mapping for routines
Setting	Production	Item – configurable item	Configures an item in the

		Target – parameter type: Item, Host, Adapter Setting – parameter name Value – parameter value	Ensemble production Makes a call to the Ens.Production:ApplySettings method
SystemSetting	Manifest	Name – class.property of the Config package Value – attribute value	Sets the values for attributes of the Config package (using the Modify method)
User	Manifest	Username – user name PasswordVar – variable containing the password Roles – list of user's roles Fullname – full name Namespace – startup namespace Routine – starting routine ExpirationDate – date after which the user will be deactivated ChangePassword – change the password upon next login into the system Enabled – whether the user is activated	Creates or modifies a user.
Var	Manifest	Name — variable name Value – variable value	Assigns a value to the variable

Variables

User-supplied variables

Some attributes can contain expressions (strings) that are expanded when the manifest is executed. There are three types of expressions that can be expanded, as follows:

- ${<Variablename>} – value of the variable (user-defined or environment variable; see below) is calculated during the execution of the manifest;
- ${#<Parametername>} – will be replaced with the value of the specified parameter from the installer's class during compilation;
- #{<CachéObjectScriptcode>} — the value of the specified Caché ObjectScript statement will be calculated during the execution of the manifest. Make sure to put quotation marks as required.

Parameter values are defined during compilation and therefore can be a part of a variable or Caché ObjectScript statement. Since variables are interpreted before the Caché ObjectScript code, you can use them in Caché ObjectScript statements, e.g: #{$ZCVT("${NAMESPACE}","L")}.

System variables

The following variables are always available:

Variable	Description	Sample value
SourceDir	(Available only when the installer is	/InterSystems/distr/

	run) Directory from which the installation (setupcache.exe or cinstall) is running.	
ISCUpgrade	(Available only when the installer is run) Indicates whether this is a new installation or an upgrade. This variable is either 0 (new installation) or 1 (upgrade).	0 (installation) 1 (update)
CFGDIR	See INSTALLDIR.	/InterSystems/Cache/
CFGFILE	Path to the CPF file	/InterSystems/Cache/cache.cpf
CFGNAME	Name of the instance	CACHE
CPUCOUNT	Number of CPU cores	4
CSPDIR	The CSP directory	/InterSystems/Cache/csp/
HOSTNAME	Name of the web server	SCHOOL15
HTTPPORT	Port of the web server	80
INSTALLDIR	Directory where Caché is installed	/InterSystems/Cache/
MGRDIR	Management directory (mgr)	/InterSystems/Cache/mgr/
PLATFORM	Operating system	UNIX
PORT	Port of the Caché superserver	1972
PROCESSOR	Name of the platform	x86-64
VERSION	Version of Caché	2015.1.1

Debugging

Sometimes it may be hard to understand what values can be assigned as node attributes' values. To figure this out, check the generated int code for the setup method. In most cases, the main call is made to the tInstaller.<ElementName> which is an object of the %Installer.Installer class which, in turn, will make direct calls to the system methods. Alternatively, you can check the code of the %Installer.<ElementName> class in which the node attributes are class properties. The program code is generated in the %OnBeforeGenerateCode, %OnGenerateCode and %OnAfterGenerateCode methods.

For debugging purposes, I recommend that you wrap a call to the installer into a transaction. For example, you can use the TSTART/TROLLBACK commands to easily undo all changes made within Caché (however, external changes, such as creating a new database file, will not be reverted).

Lastly, don't forget to set LogLevel to 3.

Examples

The MDX2JSON project provides an installer. To install the project, import the installer.xml file containing the MDX2JSON.Installer class into any namespace. You can perform import either from the SMP or by drag&dropping the file into Studio.
Then run the following command in a terminal:

```
do ##class(MDX2JSON.Installer).setup()
```

As a result, Caché will load application files from the GitHub repository and then perform installation in the default MDX2JSON namespace/MDX2JSON database, map the MDX2SJON package to %All and SAMPLES, map the ^MDX2SJON global to %All and SAMPLES create the REST application called /MDX2JSON, and so on – you will see all these steps in the terminal. For more detailed information on MDX2JSON installer, please refer to the project readme.

Even more examples

Example from the documentation.
The Sample.Installer class in the Samples namespace.
The CacheGitHubCI projects provides an installer.
The SYSMON Dashboards project provides an installer.
The DeepSee Audit project provides an installer.

Summary

%Installer is a convenient tool for distributing and deploying applications based on InterSystems Caché and Ensemble.

References

Documentation

#Caché #Deployment #Terminal #Tools #Mapping #System Administration