Article
Evgeny Shvarov · Nov 21, 2019 7m read

Open Exchange

Describing a module.xml for ObjectScript Package Manager

Hi Developers!

As discussed in previous parts of Package Manager stories to turn your IRIS application into a deployable package you just need to introduce the module.xml file into the root folder of the repository and describe all the resources.

I ' m pleased to introduce you to a new project template on Open Exchange which contains examples of how to make different types of resources of your InterSystems IRIS application a part of the ObjectScript package and so make the deployable ObjectSctipt package.

Let's see how you can describe your application resources using this template project as an example.

See the details below.

All the resources of the package are expected to be listed inside the <Module> tag of the module.xml in a form of:

<Resource Name= *"resource.name.type /">*

Let ' s go through all the types of resources in this project, where we have:

classes,

include file,

macro file,

global,

web apps.

ObjectScript Classes

First, make sure that the names of your ObjectScript classes are in order with Naming Convention and files are placed in /src/cls folder.

Read the related article on how to export ObjectScript classes in folders suitable for packaging.

e.g. If the class file sits in

```
community/objectscript/ClassExample.cls
```

the resource for it will be:

```
<Resource Name="community.objectscript.ClassExample.cls"/>
```

If you have a thousand of classes in the folder

```
/src/cls/community/objectscript/
```

The resource which describes all these classes in the folder will be:

```
<Resource Name="community.objectscript.PKG"/>
```

And when ZPM will build the package it will look for all the classes included into **community.objectscript** classes with all the sub-packages if any.

Include files with Macro definitions

Include files are expected to be placed in a /src/cls/package/include.inc order.

E.g. if you have the inc file

```
/src/inc/community/objectscript/macroexample.inc
```

You can describe it as

```
<Resource Name="community.objectscript.macroexample.INC"/>
```

Macro files

Macro files are expected in the folder:

```
/src/mac/package/macro.mac
```

if you have the mac file:

```
/src/Mac/community/objectscript/MacExample.mac
```

the resource definition will be

```
<Resource Name="community.objectscript.MacExample.MAC"/>
```

Globals

Sometimes we need to deploy the data in globals with our IRIS solution.

Export the global into the following form of files:

```
/gbl/globalname.GBL
```

and describe each in the resources.

E.g. if you have the global file as:

```
/gbl/community.objectscript.settings.GBL
```

The resource for it will be as:

```
<Resource Name="community.objectscript.settings.GBL"/>
```

And when ZPM will build a package for such a module it will look for **^community.objectscript.settings** global to include into the package.

Web Applications

There are two types of web applications in IRIS: CSP Applications and REST API applications. Let's see how they are described in ZPM.

CSP Application

If we have CSP files to deploy or just any frontend UI delivered with InterSystems IRIS we need to describe CSP Application section in module.xml.

The files in your repository for CSP Application are either csp -files, or any files suitable for the web app: js, images, html, etc.

You can place put it in any place in your repo suitable for you for development or build. The only thing this resource description does it copies files from the repo into a certain folder on the target IRIS installation.

In this example I have one csp file hello.csp which sits in:

```
/src/csp/hello.csp
```

CSP application is described with <CSPApplication> tag inside <Module> tag.

Suppose we want to deploy a web application /hello which will contain hello.csp. here is the deployment tag for this:

```
<CSPApplication
        Url="/hello"
        SourcePath="/src/csp"
        DeployPath="${cspdir}hello"
        ServeFiles="1"
        Recurse="1"
        CookiePath="/hello"
        UseCookies="2"
        MatchRoles=":${dbrole}"
        PasswordAuthEnabled="1"
        UnauthenticatedEnabled="0"
       />
```

Here Url parameter is the name of the web app you create on the target server - **/hello** in this case.

the SourcePath parameter relates to the path to files you want to copy to target the InterSystems IRIS server.

The DeployPath parameter is used to set the target folder on the remote server.

You can use {$cspdir} variable here to point the deployment under the regular IRIS CSP folder.

Cookie path is meaningful for authorization and sessions

Use MatchRoles to provide the security resource you want the CSP Application to have access to.

The presented example CSPApplication tag will be deployed as a web app which will be accessible on:

```
<server:port>/hello/hello.csp
```

And hello.csp itself is a very basic example that returns the current day using ObjectScript with **#()#** syntax and uses **<server>** CSP tag to call the ObjectScript class method.

The result you should see.

REST-API application

REST API application

Another type of web application you may want to deploy is a RESTFul app when we setup the Url and ObjectScript dispatch class which contains the REST API map to dispatch URL calls into ObjectScript method calls to process and return data (typically JSON data).

Suppose you want to deploy /rest-test app with community.objectscript.RESTExample.cls as a dispatch class. Here is the tag description:

```
<CSPApplication
        Url="/rest-test"
        Recurse="1"
        MatchRoles=":${dbrole}"
        PasswordAuthEnabled="1"
        UnauthenticatedEnabled="0"
        DispatchClass="community.objectscript.RESTExample"
        ServeFiles="1"
        CookiePath="/rest-test"
        UseCookies="2"
       />
```

The description of the REST API CSP application is almost similar the key difference is that parameter DispatchClass contains the name of the class which will dispatch REST-API calls.

DispatchClass="community.objectscript.RESTExample"

The whole text of this example module.xml is:

Spoiler

How to test it?

Install zpm-client or just build the IRIS container in this repo - this dockerfile will run IRIS and will install ZPM.

Once you have zpm installed, you can test how this sample works with the following command:

```
USER>zpm

Zpm:USER>install objectscript-package-template
```

It will install classes from community.objectscript package, inc and mac files, one global ^community.objectscript.settings and two applications: /hello and /rest-test.

You are very welcome to use this sample template to play with ZPM packages or/and build your own and submit it to the Community ObjectScript Package Manager Registry or use it for your own registry.

How to test your own package before publishing in Open Exchange or in production Registry

Before publishing your package you want to check if the module.xml works properly. With ZPM you can build the package before publishing.

Consider you have similar to this project structure and have IRIS running in docker. Then you can do the following:

1. Load the package. Load command will load the module.xml from the folder and resources described in module.xml

```
IRISAPP>zpm

zpm:IRISAPP>load /irisdev/app
```

If there are no errors during load you can test the package with the following command:

```
zpm:IRISAPP>your-package-name package -v
```

e.g.:

```
zpm:IRISAPP>objectscript-package-template package -v
```

Once you don't see any errors the package is formed well.

You also can build a test-registry (as described here), publish your package there and install it with ZPM client and check if it does what should during the installation.

Stay tuned for the next ObjectScript Package Manager stories!

#Deployment #ObjectScript #InterSystems Package Manager (IPM) #InterSystems IRIS #Open Exchange

Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/describing-modulexml-objectscript-package-manager