Artículo

[Alberto Fuentes](#) · 17 dic, 2021 Lectura de 6 min

[Open Exchange](#)

Estructura del módulo ZPM: Cómo empaquetar tu solución de InterSystems

¡Hola desarrolladores!

Hace un tiempo lanzamos [InterSystems Package Manager](#) - ZPM. Uno de los objetivos de ZPM es permitir empaquetar soluciones y enviarlas al registro de ZPM para que su implementación sea tan sencilla como el comando "install tu-paquete".

Para ello es necesario introducir el archivo module.xml en tu repositorio, que describe en qué consiste tu paquete de InterSystems IRIS.

En este artículo se describen las diferentes partes de module.xml, que te ayudarán a configurar el tuyo.

Comenzaré por el paquete samples-objectscript, que instala en IRIS la [aplicación Sample ObjectScript](#) y se puede instalar mediante:

```
zpm: USER>install samples-objectscript
```

Este es probablemente el paquete más sencillo y aquí está el archivo [module.xml](#), que describe el paquete:

```
<?xml version="1.0" encoding="UTF-8"?>
<Export generator="Cache" version="25">
  <Document name="samples-objectscript.ZPM">
    <Module>
      <Name>samples-objectscript</Name>
      <Version>1.0.0</Version>
      <Packaging>module</Packaging>
      <SourcesRoot>src</SourcesRoot>
      <Resource Name="ObjectScript.PKG"/>
    </Module>
  </Document>
</Export>
```

Vamos a revisar el documento detalladamente.

```
<Export generator="Cache" version="25">
```

El archivo module.xml pertenece a la familia de documentos de Cache/IRIS xml, por lo que en esta línea se establece la relación para que las librerías internas reconozcan el documento.

La siguiente sección es <Document>

```
<Document name="samples-objectscript.ZPM">
```

Tu paquete debe tener un nombre. El nombre puede contener letras en minúscula y el signo "-". Por ejemplo, samples-objectscript en este caso. Coloca el nombre de tu paquete en la cláusula name de la etiqueta Document con la extensión .ZPM.

Los elementos internos de Document son:

<Name> - el nombre de tu paquete. En este caso:

```
<Name>samples-objectscript</Name>
```

<Version> - la versión del paquete. En este caso:

```
<Version>1.0.0</Version>
```

<Packaging>module</Packaging> - el tipo de paquete. Coloca aquí el parámetro del módulo.

```
<Packaging>module</Packaging>
```

<SourcesRoot> - una carpeta, donde ZPM buscará a ObjectScript para su importación.

En este caso, te aconsejamos que busques ObjectScript en la carpeta /src:

```
<SourcesRoot>src</SourcesRoot>
```

<Resource Name> - los elementos de ObjectScript que se importarán. Pueden ser paquetes, clases, inclusiones, globals, dfi, etc.

La estructura en la carpeta SourceRoot puede ser la que se genera con el VS Code al trabajar con ObjectScript por defecto.

Aunque también puedes utilizar la versión clásica que tenía esta forma:

/cls - todas las clases de ObjectScript en formato Folder=Package, Class=file.cls. Los subpaquetes son subcarpetas.

/inc - todos los archivos incluidos en formato file.inc.

/mac - todas las rutinas en mac.

/int - todas las rutinas "intermedias" (alias "otro" código, el resultado de una compilación de código mac, u ObjectScript sin clases ni macro).

/gbl - todos los globals en formato xml de exportación.

/dfi - todos los archivos DFI en formato xml de exportación. Cada tabla dinámica se incluye en el archivo pivot.dfi, y todos los paneles de control se incluyen en el archivo dashboard.dfi.

Por ejemplo, aquí importaremos la página de ObjectScript. Esto indicará a ZPM que busque la carpeta

/src/cls/ObjectScript y que importe todas las clases procedentes de ella:

```
<Resource Name="ObjectScript.PKG" />
```

Así que para preparar tu solución para el empaquetado, coloca las clases de ObjectScript en alguna carpeta de tu repositorio dentro de la carpeta /cls y pon todos los paquetes y clases en package=folder, class=file.cls.

Rutinas para empaquetar con mac

Esto es muy similar a las clases. Basta con colocar las rutinas en la carpeta /mac. [Por ejemplo:](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<Export generator="Cache" version="25">
  <Document name="DeepSeeButtons.ZPM">
    <Module>
      <Name>DeepSeeButtons</Name>
      <Version>0.1.7</Version>
      <Packaging>module</Packaging>
      <SourcesRoot>src</SourcesRoot>
      <Resource Name="DeepSeeButtons.mac" />
    </Module>
  </Document>
</Export>
```

Otros elementos

También hay elementos opcionales como:

```
<Author>
```

Que pueden contener elementos como <Organization> y <CopyrightDate>.

Por ejemplo:

```
<Author>
  <Organization>InterSystems</Organization>
  <CopyrightDate>2019</CopyrightDate>
</Author>
```

Empaquetado de aplicaciones CSP/Web

ZPM también puede implementar aplicaciones web.

Para que funcione, introduce el elemento CSPApplication con las cláusulas de los [parámetros de la aplicación CSP](#).

Por ejemplo, observa la etiqueta CSPApplication de [module.xml](#) en DeepSeeWeb:

```
<CSPApplication
  Url="/dsw"
  DeployPath="/build"
  SourcePath="{cspdir}/dsw"
  ServeFiles="1"
```

```
Recurse="1"  
CookiePath="/dsw"  
</>
```

Esta configuración creará una aplicación web con el nombre /dsw y copiará todos los archivos de la carpeta /build del repositorio a la carpeta \${cspdir}****/dsw, una carpeta que se encuentra debajo del directorio IRIS csp.

Aplicación API REST

Si se trata de una aplicación API-REST, el elemento CSPApplication contendrá la clase dispatch y podría ser similar al módulo MDX2JSON.xml:

```
<CSPApplication  
  Path="/MDX2JSON"  
  Url="/MDX2JSON"  
  CookiePath="/MDX2JSON/"  
  PasswordAuthEnabled="1"  
  UnauthenticatedEnabled="1"  
  DispatchClass="MDX2JSON.REST"  
</>
```

Dependencias

Tu módulo podría esperar la presencia de otro módulo que esté instalado en el sistema de destino. Esto se podría describir mediante el elemento <Dependencies> situado dentro del elemento <Document>, que podría contener varios elementos <ModuleReference>, cada uno de los cuales tiene <Name> y <Version> y que indican qué otros módulos, con qué versión deberían instalarse antes que la tuya. Esto hará que ZPM verifique si los módulos están instalados y, en caso contrario, realice la instalación.

Este es un ejemplo de dependencia del módulo DSW en el módulo MDX2JSON:

```
<Dependencies>  
  <ModuleReference>  
    <Name>MDX2JSON</Name>  
    <Version>2.2.0</Version>  
  </ModuleReference>  
</Dependencies>
```

Otro [ejemplo](#), donde [ThirdPartyPortlets](#) depende de [Samples BI](#) (holefoods):

```
<Dependencies>  
  <ModuleReference>  
    <Name>holefoods</Name>  
    <Version>0.1.0</Version>  
  </ModuleReference>  
</Dependencies>
```

Además hay opciones para ejecutar tu código de forma arbitraria para configurar los datos y el entorno, pero hablaremos sobre esto en los siguientes artículos.

Cómo crear tu propio paquete

¡Muy bien! Una vez que tengas un module.xml, puedes intentar crear el paquete y probar si la estructura de module.xml es acertada.

Puedes probarlo por medio del cliente zpm. Instala ZPM en un sistema IRIS y carga el código del paquete con el comando load:

```
zpm: NAMESPACE>load path-to-the-project
```

La ruta indica la carpeta que contiene los recursos correspondientes para el paquete y tiene el module.xml en la carpeta raíz.

Por ejemplo, puedes probar el paquete creando [este proyecto](#). Compruébalo y crea un contenedor con docker-compose-zpm.yml.

Abre el terminal en el namespace SAMPLES y llama a ZPM:

```
zpm: SAMPLES>
```

```
zpm: SAMPLES>load /iris/app
```

```
[samples-objectscript] Reload START
[samples-objectscript] Reload SUCCESS
[samples-objectscript] Module object refreshed.
[samples-objectscript] Validate START
[samples-objectscript] Validate SUCCESS
[samples-objectscript] Compile START
[samples-objectscript] Compile SUCCESS
[samples-objectscript] Activate START
[samples-objectscript] Configure START
[samples-objectscript] Configure SUCCESS
[samples-objectscript] Activate SUCCESS
```

La ruta es "/iris/app" porque en docker-compose-zpm.yml indicamos que mapeamos la raíz del proyecto en la carpeta /iris/app en el contenedor. Por tanto, podemos utilizar esta ruta para indicar a ZPM desde dónde cargar el proyecto.

¡El proyecto se cargó correctamente! Esto significa que module.xml podría utilizarse para enviar un paquete al repositorio de la Comunidad de Desarrolladores.

Ahora ya sabes cómo crear un archivo module.xml adecuado para tu aplicación.

Cómo enviar la solicitud al repositorio de la Comunidad de InterSystems

1. Tu aplicación debe estar en [Open Exchange](#)
2. Al dar de alta tu aplicación, simplemente marca el checkbox Publish in Package Manager. A continuación simplemente espera a que pase el proceso de aprobación para que se publique.

¡Y deberías tener un module.xml funcionando correctamente!

[#InterSystems Package Manager \(IPM\) #Principiante #Tutorial #InterSystems IRIS #Open Exchange](#)
[Ir a la aplicación en InterSystems Open Exchange](#)

URL de
fuente: <https://es.community.intersystems.com/post/estructura-del-m%C3%B3dulo-zpm-c%C3%B3mo-empaquetar-tu-soluci%C3%B3n-de-intersystems>