Cómo crear un módulo nuevo

Un módulo necesita 3 archivos básicos:

- **init.php**: Página que se carga al lanzar el módulo.
- **config.xml**: Fichero de configuración con información sobre el módulo: nombre, descripción, permisos, etc.
- **MiModulo.class.php**: Clase utilizada para implementar la comunicación entre los módulos., donde *MiModulo* será el nombre del módulo (sensible a mayúsculas).

Estructura básica de estos ficheros:

config.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<module name="MiModulo">
          <screenname lang="en">My Module</screenname>
          <screenname lang="es">Mi Modulo</screenname>
</module>
```

Donde <*module name=“MiModulo“*> indica el nombre interno que tendrá en el sistema el módulo y <*screenname lang="XX"*>*Mi Modulo*<*/screenname*> será el nombre que aparecerá en pantalla para el idioma *XX*.

MiModulo.class.php

El nombre del fichero debe coincidir con el nombre interno del módulo. Este es el contenido mínimo que debe tener este fichero.

```
<?php
include_once('core/modules/IJamesModule.class.php');
class MiModulo extends IjamesModule
{
          function MiModulo($myself)
          {
          }
}
?>
```

init.php

El contenido de este fichero será el que indique el comportamiento del módulo en *php*. Su estructura básica será la de un archivo *php*, pero inicializando algunos valores. Un primer contenido sería:

```
<?php
          include_once(“core/html/HtmlInterface.class.php”);
          $html = new HtmlInterface(“MiModulo”);
          // Comprueba que se puede ejecutar el Módulo
          // Establece el contexto de Ejecución(Usuario y Grupo
          // actual, ...)
?>
<html>
<head>
<title>Mi Modulo</title>
```

```
</head>
<body>
          ¡Hola mundo!
</body>
</html>
```

Instalación del módulo

Todos los ficheros que forman parte del módulo se empaquetan en un archivo comprimido que debe tener el mismo nombre que el módulo que contienen. En el caso de este ejemplo, sería *MiModulo.zip*. La estructura interna de este fichero comprimido es la siguiente:

Directorio

/MiModulo

Archivos

/MiModulo/config.xml

/MiModulo/MiModulo.class.php

/MiModulo/init.php

...cualquier otro archivo que se desee incluir, siempre dentro del directorio /MiModulo

Los módulos se instalan el sistema utilizando el *Módulo de Instalación/Elminación de módulos.* En dicho módulo hay que indicarle el archivo *MiModulo.zip* y seguir los pasos de la instalación. Esto registrará el módulo en el sistema. A partir de este momento ya se puede añadir el módulo a cualquier grupo del sistema.

Desarrollando el módulo

Veamos ahora algunas cuestiones básicas en el desarrollo de un módulo.

El código del módulo se desarrollará en el archivo *init.php* o en archivos incluidos por éste (código php), mientras que los valores de configuración (como los permisos) que deban ser tenidos en cuenta en el sistema hay que incluirlos en el archivo de configuración (código xml).

Usuario Activo

Para acceder al usuario activo del sistema se hace mediante:

```
$GLOBALS[CURRENT_USER]
```

Esto devolverá un objeto de la clase *User* representando al usuario activo en el sistema en ese momento.

Algunas funciones de la clase *User* son:

```
$GLOBALS[CURRENT_USER]->GetLogin(); // Devuelve el nombre
$GLOBALS[CURRENT_USER]->GetId(); // Devuelve el Identificador
```

Un ejemplo de la utilización de esto sería: (en *init.php*)

```
<?php
          include_once("core/html/HtmlInterface.class.php");
          $html = new HtmlInterface("MiModulo");
?>
<html>
<head>
<title>Mi Modulo</title>
</head>
<body>
<?php
          echo "El login es: ".$GLOBALS[CURRENT_USER]->GetLogin();
          echo "<br>";
          echo "El id es: " . $GLOBALS[CURRENT_USER]->GetId();
          echo "<br>";
?>
</body>
</html>
```

Grupo Actual

La forma de obtener el grupo actual es a través de otra variable del sistema:

```
$GLOBALS[CURRENT_GROUP]
```

Esto nos devolverá un objeto de la clase *Group*. Algunos métodos de los objetos de esta clase son:

```
$GLOBALS[CURRENT_GROUP]->GetName(); // Devuelve el nombre
$GLOBALS[CURRENT_GROUP]->GetId(); // Devuelve el Identificador
```

Ejemplo (dentro de init.php)

```
<?php
```

```
          include_once("core/html/HtmlInterface.class.php");
          $html = new HtmlInterface("MiModulo");
?>
<html>
<head>
<title>Mi Modulo</title>
</head>
<body>
<?php
          echo "El grupo es" . $GLOBALS[CURRENT_GROUP]->GetName();
          echo "<br>";
          echo "El id es: " . $GLOBALS[CURRENT_GROUP]->GetId();
          echo "<br>";
?>
</body>
</html>
```

Permisos

Los permisos para que puedan ser configurarse en el sistema utilizando los roles deben haber sido primero definidos en el archivo de configuración, *config.xml,* antes de instalar el módulo. Veamos un ejemplo como añadir un permiso a dicho archivo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<module name="MiModulo">
   <screenname lang="es">Mi Modulo</screenname>
   <permissions>
      <permission name="miPermiso">
           <screenname lang="es">Mi permiso</screenname>
           <description lang="es">
               Permiso de prueba para MiModulo
           </description>
      </permission>
   </permissions>
</module>
```

Comentario:

La etiqueta <*permissions*> indica que vamos a comenzar a definir permisos (uno o varios). Para cada permiso hay que definir como mínimo el nombre interno:

<*permission name="miPermiso"*>

que será el que se utilice en el desarrollo del módulo.

La etiqueta *screenname* indica el nombre 'amigable' que se mostrará en pantalla y la etiqueta *description*, la descripción que aparecerá en pantalla explicando la utilidad de dicho permiso.

Consulta de permisos

Para consultar los permisos de un usuario en el código de un módulo utilizaremos la función: *HasPermission* de la clase *User*. Un ejemplo de uso es el siguiente:

```
<?php
          if ($GLOBALS[CURRENT_USER]->HasPermission("miPermiso"))
```

```
                    echo "El usuario tiene el permiso 'miPermiso'";
          else
                    echo "No tienes el permiso 'miPermiso'";
?>
```

donde *miPermiso* en la llamada a la función HasPermission será el nombre que tenga el permiso en el archivo *config.xml* (sensible a mayúsculas y minúsculas).

Opciones de Configuración

Un Módulo puede utilizar variables de configuración (por ejemplo, el tamaño máximo de los archivos que se pueden subir a un repositorio). Estas variables se definirán en el archivo *config.xml* (antes de instalar el módulo), se modificarán en el módulo de *Configuración* y se podrán consultar desde el código del módulo.

Definición de Opciones de Configuración en *config.xml*

```
<module name="MiModulo">
...
  <config>
     <!-- Primero definiremos una clave de configuración para un valor
          entero -->

      <configkey name="size" type="integer" level="system">

      <!-- name="size" será el nombre que utilice en el sistema,
           type="integer" indica que es de tipo entero (otros tipos serán
           "boolean", "option" o "string") y  level="system" que será una
           variable configurable para todo el sistema (podría serlo para
           "group" o  "user") -->

         <screenname lang="es">Tamaño</screenname>
         <description lang="es">
            Tamaño máximo asignado por usuario
         </description>
      </configkey>

     <!-- Ahora vamos a definir una clave de configuración con una lista
          de valores opcionales -->

      <configkey name="format" type="option" level="system">
         <screenname lang="es">Formato</screenname>
         <description lang="es">Formato de envío</description>
         <options>
            <option name="text">
                 <screenname lang="es">Texto</screenname>
            </option>
            <option name="html">
                 <screenname lang="es">HTML</screenname>
            </option>
         </options>
      </configkey>
   </config>
...
</module>
```

Consulta de Valores de Configuración en el Módulo

Para consultar los valores de configuración utilizaremos la función:
GetConfig(nombre, valor_por_defecto) de la clase Module

donde *nombre* será el nombre de que se le haya dado a la clave en el archivo config.xml (sensible a mayúsculas y minúsculas) y *valor_por_defecto* será un valor que pasemos por defecto a la función por si no se le ha asignado aún un valor a dicha clave en el sistema.

Ejemplo:

```
<?php
          echo "El tamaño asignado es: ";
          echo $GLOBALS[CURRENT_MODULE]->GetConfig("size", 5000);
?>
```

Obtener Información de otros Usuarios/Grupos distintos al Actual

Utilizaremos:

```
$GLOBALS[JUGRSYSTEM]
```

Ejemplos:

```
// Obtener un usuario
$user = $GLOBALS[JUGRSYSTEM]->GetUser("admin");
$user->GetId();

// Obtener un grupo
$group = $GLOBALS[JUGRSYSTEM]->GetGroup(1);
$group->GetName();
```

Acceso a Base de Datos

Utilizaremos la clase *DBJames*, que nos servirá como capa de abstracción para las funciones de acceso a bases de datos de *PHP*.

Funciones y Atributos

- *Query(miConsulta):* Realiza la consulta miConsulta sobre la base de datos y se posiciona sobre el primer registro.
- *row[miCampo]:* Accede al campo miCampo del registro actual.
- *Next():* Accede al siguiente registro.
- *EOF:* Devuelve un booleano indicando si hemos sobrepasado el último registro.

Ejemplo típico de uso:

```
...
$dbObj = new DBJames();
$dbObj->Query("SELECT * FROM ModNews ORDER BY date DESC where idG=2");
while(!$dbObj->EOF)
{
          echo "<h1>";
```

```
            echo $dbObj->row[ "title" ];
            echo "<h1>";
            echo "<hr>";
            echo $dbObj->row[ "body" ];
            echo "<br>";
            $dbObj->Next();
}
...
```