# GUÍA DE PROGRAMACIÓN DEL COMPONENTE GENERADOR DE EXCEL

LUNES, 28 DE DICIEMBRE DE 2009

## CONTENIDO

## INTRODUCCIÓN

Este componente permite exportar a un fichero de Microsoft Excel (xls) una colección de datos. El componente iterará por la colección creando una fila por cada elemento, y una columna por cada atributo del objeto que se desee exportar. Las columnas se especificarán mediante un mapeo entre el nombre del atributo del objeto y el título que se quiere dar a la columna. Por ejemplo, para una lista de objetos de tipo Persona, siendo "name" un atributo de esa clase que se desea exportar a la columna nombre, el mapeo se especificaría de la siguiente forma: "name=Nombre".

El componente retornará un ByteArrayOutputStream con el fichero, de manera que puede ser utilizado tanto en aplicaciones J2EE (web) como stand-alone (J2SE), siendo responsabilidad del código cliente qué hacer con él (ej.: guardarlo a fichero, serializarlo sobre la response, etc...).

La API es muy sencilla:

```
public ByteArrayOutputStream createExcelFile(List<?> data, String
mapColumns).
```

Donde el formato de mapColumns es como sigue: att1=colHeader1; att2=colHeader2;... ; attN=colHeaderN.

El mapeo no es obligatorio. Si no se especifica, la exportación contendrá todos los atributos del bean, siendo la cabecera de la columna el nombre del atributo.

Como se ha indicado, el componente puede ser utilizado de manera independiente. No obstante aquí se precisará como integrarlo y utilizarlo en aplicaciones web construidas según la arquitectura.

## MODELO

El único requisito que debe satisfacer el modelo es contener una lista de objetos (ej.: List<Clientes> listaClientes), que contendrá los datos a partir de los cuales generar el fichero Excel.

También debe contener un campo para que el comando pueda almacenar el resultado de la ejecución. Este puede ser de tipo byte[], ByteArrayOutputStream o bien de tipo FileTransferItem, que es la abstracción que utiliza la arquitectura para la gestión de ficheros y que permitirá usar el componente de descarga proporcionado por la arquitectura.

## CONTROLADOR

Siguiendo la normativa de Arquitectura, el controlador extenderá de BaseBackingBean parametrizado con el tipo del modelo definido.

También según la arquitectura, los comandos no tiene valor de retorno, sino que el resultado del comando se deposita en el parámetro de entrada del método execute (es decir, el modelo). Por tanto, el comando, que será el encargado de generar el fichero Excel, dejará el resultado de la exportación en el comando. El controlador deberá recoger este valor tras la ejecución. Para ello es necesario sobrescribir el método processResponse (o definir el delegate apropiado). En el processResponse, se podrá acceder al modelo y por tanto al fichero.

La arquitectura implementa la gestión de descarga de ficheros, a través del delegate "fileSupport". Éste ofrece un método "downloadFile" que serializará el objeto sobre la response, de manera que al usuario se le mostrará desde el navegador el diálogo para Abrir/Guardar el fichero. Este método utiliza como abstracción para tratar los ficheros el objeto de tipo FileTransferItem. Por tanto, es preciso crear un objeto de este tipo (bien en el modelo, por lo que se creará en la ejecución del comando, bien directamente en processResponse).

A continuación se muestra un ejemplo de cómo el fichero que se encuentra en memoria puede ser manipulado para que el usuario pueda descargarlo y visualizarlo.

```
@Override

protected void processResponse(String whichCommand) {

          if (getModel().getFile() != null) {

                getFileSupport().downloadFile(getModel().getFile());
```

```
            }


}
```

## FICHERO FACES-CONFIG.XML

La configuración que se tiene que realizar en el fichero faces-config.xml es la genérica para una aplicación siguiendo la normativa de Arquitectura.

Para poder utilizar la funcionalidad proporcionada por FileSupport, es necesario configurar el delegate fileSupport sobre el controlador. Para ello basta con especificar el componente de soporte a ficheros que ofrece la arquitectura (#{defaultFileSupport}).

```
<managed-bean>

  <managed-bean-name>FlowCreateExcel</managed-bean-name>

  <managed-bean-class>

       excel.jsf.FlowCreateExcel

  </managed-bean-class>

  <managed-bean-scope>request</managed-bean-scope>

  <managed-property>

         <property-name>fileSupport</property-name>

           <property-class>

          filetransfer.FileSupport

      </property-class>

           <value>#{defaultFileSupport}</value>
```

```
        </managed-property>

</managed-bean>
```

Nota: Es necesario confirmar que se ha realizado el mapeo a CustomFacesVariableResolver, esta clase se encarga de resolver la instancia de FileSupport.

```
<application>

  <variable-resolver>

           jsf.resolver.CustomFacesVariableResolver

  </variable-resolver>

 </application>
```

## CONFIGURACIÓN DE SPRING, FICHERO APPLICATIONCONTEXT.XML

En el fichero applicationContext.xml, solamente será necesario definir y configurar el Bean del comando, inyectándole el componente de exportación a Excel que define la arquitectura bajo el nombre "**excelGenerator**", de tipo ExcelGenerator.

Opcionalmente, el comando puede definir otra propiedad para los mapeos entre el objeto a exportar y las columnas.

```
<bean id="commandCreateExcel"

      class="excel.command.CommandCreateExcel">
```

```
    <property name="excel" ref="excelGenerator"></property>

    <property name="mappings"

              value="dni=DNI;nombre=NOMBRE;apellidos=APELLIDO"/>

</bean>
```

## COMANDOS

La Clase del comando tiene que seguir la normativa de Arquitectura, extendiendo AbstractCommand parametrizado con el tipo del modelo.

Esta Clase necesita contener una instancia del componente de Excel. La inyección de esta instancia se realizara en Spring. El comando únicamente debe definir el atributo de tipo ExcelGenerator y el método setter (opcionalmente el atributo para los mappings).

```
public class CommandCreateExcel extends AbstractCommand<ModelExcel>  {

        protected Log logger = LogFactory.getLog(this.getClass());

        private  ExcelGenerator excel;

        private  String mappings;

   . . .

   public void setExcel(ExcelGenerator e) {

      this.excel = e;

   }



   public void setMappings(String s) {
```

```
        this.mappings = s;

    }

}
```

El comando utilizará el generador de excel inyectado por Spring, invocando su método createExcelFile proporcionando la colección (obtenida del modelo) y la lista de mapeos, obteniendo un ByteArrayOutputStream con el fichero xls.

Para integrar con la funcionalidad de descarga de ficheros, se creará un objeto de tipo FileTransferItem utilizando la implementación DefaultFileTransferItem, el cual recibirá como parámetros de entrada del fichero: el nombre, el byte Array y el contentType, éste último parámetro de entrada en el caso de un fichero Excel es “application/octet-stream”. Este nuevo objeto se añadirá al modelo, por lo que quedará disponible al controlador para tratarlo en processResponse.

```
@Override

protected void doExecute(ModelExcel commandData) {

            ByteArrayOutputStream baos =

                    excel.createExcelFile(commandData.getClientes(),

                    this.mappings);

            FileTransferItem fti = new DefaultFileTransferItem

                                                    ("ClienteTest.xls",

                                                    baos.toByteArray(),

                                                "application/octet-stream");
```

```
            commandData.setFile(fti);

      }
```

## VISTAS

Desde la perspectiva de las vistas la exportación a excel no es más que la ejecución de un comando más. Por tanto, solamente será necesario invocar el método execute() del controlador e indicarle el nombre del comando utilizando el Tag ejemplo:command.

```
<h:commandLink action="#{FlowCreateExcel.execute}"

      value="Create Excel" >

     <ejemplo:command name="commandCreateExcel">

</ejemplo:command>

</h:commandLink>
```

## PROPIEDADES DE CONFIGURACIÓN

El componente de exportación a Excel, definido a nivel de arquitectura, ofrece una serie de propiedades que permiten configurar el formato del fichero obtenido, y pueden ser configuradas como propiedades de la aplicación en el fichero "application.properties". A continuación se muestran las propiedades (entre paréntesis su valor por defecto):

- fcm.excel.border (1) => Ancho del borde de la cabecera
- fcm.excel.fontColorHeader ("209,0,0") => Color de fuente para la cabecera
- fcm.excel.adjust (true) => Determina si autoajustar las celdas.
- fcm.excel.totalRowsPerSheet (65534) => Número máximo de filas en una hoja.
- fcm.excel.sizeFont (10) => Tamaño de fuente
- fcm.excel.fontName ("Arial") => Tipo de fuente