Guía para el desarrollador

Canal Web

Fecha:<04/11/2010>

Instituto de Empresa

< Canal Web>

1 Información de Documento

Título del Documento:	**Aproximación Funcional**
Nombre del Fichero:	IE-Guia para el desarrollador_v1.3.docx
Autor / es:	
Equipo:	

1.1 Revisión del Documento

Versión	**Fecha**	**Cambiado por**	**Cambio**
0.1	*19/10/2010*		*Versión inicial.*
1.2	*04/11/2010*	*Accenture*	*Intregación contenidos*
1.5	*17/12/2010*	*Accenture*	*Apartado 3.2.8, 4 y 10*

1.2 Aprobación del Documento

Revisado por:

<Nombre del revisor/es>

Nombre Fecha

Aprobado por:

<Nombre del aprobador/es>

Nombre Fecha

Tabla de Contenidos

2 Introducción

El objetivo de este documento es definir el entorno de trabajo de los desarrollos de portales con Vignette.

Se tratarán temas como la instalación de las herramientas de desarrollo en local (Eclipse), la estructura de los proyectos y componentes a crear, tanto en Eclipse como en Vignette, así como su posterior despliegue en los diferentes entornos.

También se presentarán otro tipo de políticas y procedimientos relativas a la gestión de versiones (etiquetado de versiones, branching,…).

Es sin duda, un documento orientado a los nuevos desarrolladores, con el fin de serviles como referencia en su trabajo diario.

En algunos apartados también se incluye una referencia al manual oficial de Vignette para consultar en caso de necesitar una mayor información.

3 Herramientas de Desarrollo

3.1 Instalación de Java 6

El primera paso a la hora de montar el entorno de desarrollo en local, es descargar la JDK, en este caso la versión 6, desde http://www.oracle.com/technetwork/java/javase/downloads/index.html

Se selecciona la plataforma donde se instalará y se aceptan las condiciones.

Se instalará la maquina virtual y los jar necesarios del Java Development Kit en la ruta por defecto (se puede escoger otra ruta).

A continuación será necesario crear una variable de entorno de sistema con nombre "*JAVA_HOME*" cuyo valor será la ruta en la cual se ha instalado el JDK ej. "*C:\Program Files\Java\jdk1.6.0_22*".

3.2 Instalación y Configuración del IDE Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto, multiplataforma para el desarrollo de "Aplicaciones de Cliente Enriquecido".

IDE Eclipse será el entorno de desarrollo para los portales del Instituto de Empresa.

Se usará la ultima versión disponible de esta herramienta: Eclipse "*Helios*". La ruta donde descargar el software es: http://www.eclipse.org/downloads/

A continuación se procede a descomprimir el archivo descargado en una ruta cualquiera.

Una vez instalado se procederá a realizar una configuración mínima a través de los siguientes pasos:

3.2.1 Configurar Eclipse: Definir el Workspace

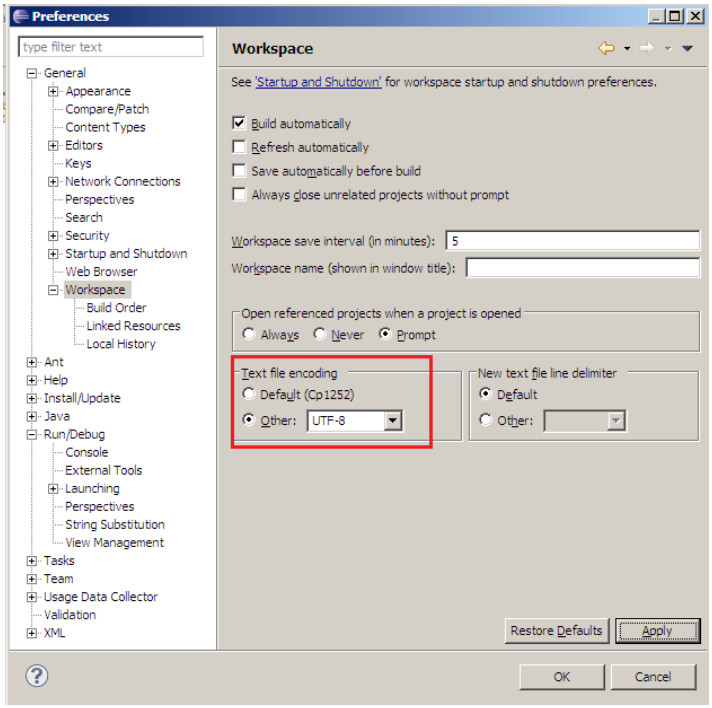
El workspace o entorno de trabajo, es un directorio a partir del cual el desarrollador, creará los proyectos a desarrollar. En él además se guarda información relativa a la personalización que realiza el usuario de la herramientas (vistas,…)

Cuando se accede a Eclipse por primera vez, la herramienta solicita introducir la ruta donde del workspace, para ello, antes de anda, será necesario crear una carpeta "*D:\DevIE*".

Es importante que todos los desarrolladores utilicen la misma carpeta para que puedan compartir ficheros de configuración sin problemas.

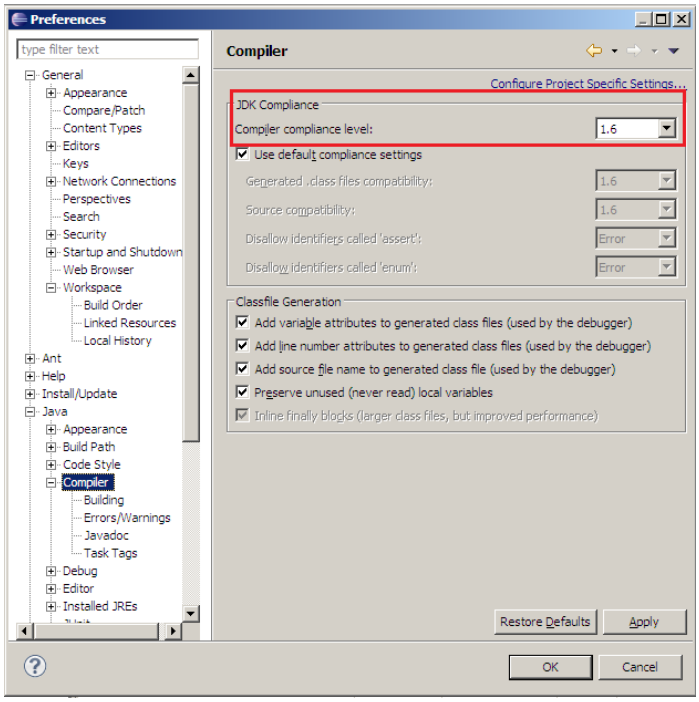
Ejecutar el fichero "eclipse.exe" y definir el workspace apuntando a la carpeta que indicada anteriormente.

3.2.2 Configurar Eclipse: Text File Encoding



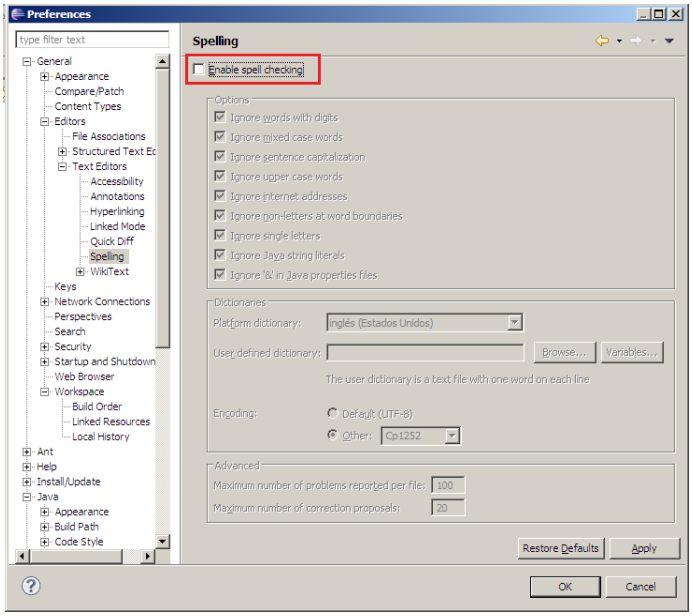
En el IDE ir a la opción "Window->Preferences->General->Workspace->Text file encoding" y escoger la opción "**Other:UTF-8**"

3.2.3 Configurar Eclipse: Compiler



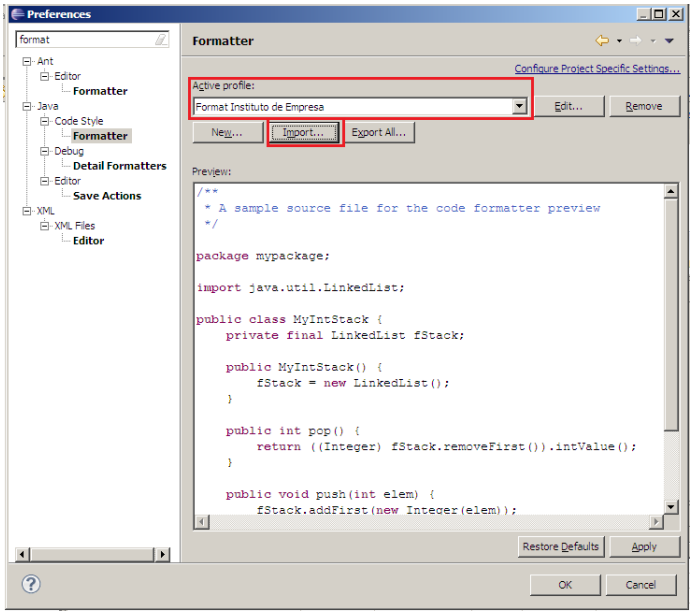
En el IDE ir a la opción "*Window->Preferences->Java->Compiler->Compiler compliance level*" y escoger la opción "**1.6**"

3.2.4 Configurar Eclipse: Editor



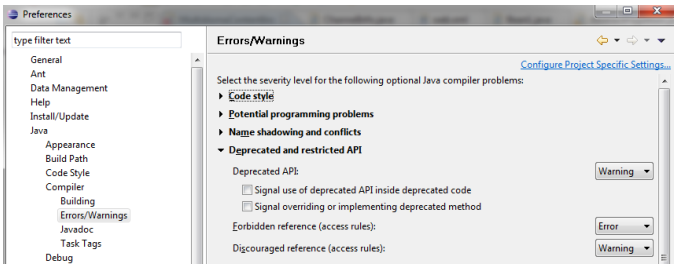
En el IDE ir a la opción "Window->Preferences->General->Editors->Text Editors->Spelling" y desactivar la opción "Enable spell checking"

3.2.5 Configurar Eclipse: Formateo de código



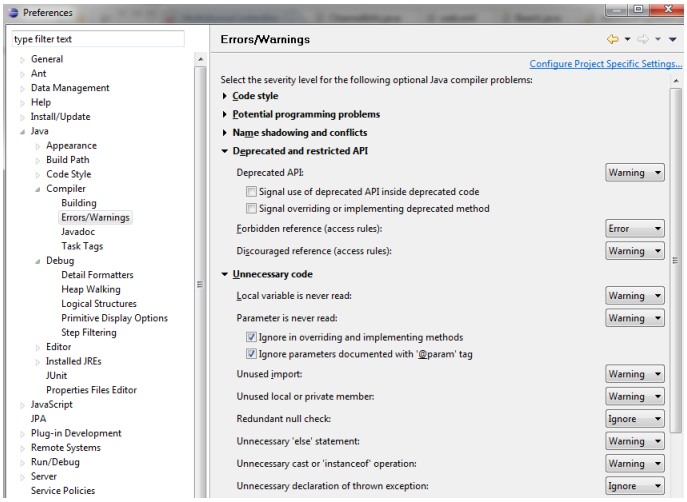
En el IDE ir a la opción "Window->Preferences->Java > Code Style > Formatter" importar la configuración de formateo de código del Instituto de Empresa, que se encuentra en el fichero eclipse_formatIE.xml

3.2.6 Configurar Eclipse: Warnings & Error



En el IDE ir a la opción "Window->Preferences->Java > Compiler > Errors/Warning" para asegurarse que el uso de métodos deprecados de la API de Java genera un aviso:

Deprecated API → Warning



También será conveniente que Eclipse avise cuando se está declarando una variable local que nunca es leía, cuando se están importando paquetes innecesarios, cuando se está pasando un parámetro que no se lee nunca, cuando se crea un método privado que no se usa,…

En general todo aquello que ayude a tener un código más limpio y por lo tanto más fácil de comprender y mantener.

3.2.7 Configurar Eclipse: Instalación de la herramienta de despliegue

Como se comentará más adelante en el apartado correspondiente: "Deploy", la herramienta seleccionada para hacer el deploy de las aplicaciones es Ant.

Además de las ventajas propias de la herramienta, a nivel de configuración hay que destacar que Eclipse ya incluye el plugin de Ant, por lo que no es necesario realizar ninguna instalación adicional.

3.2.8 Configurar Eclipse: Instalación del gestor de versiones

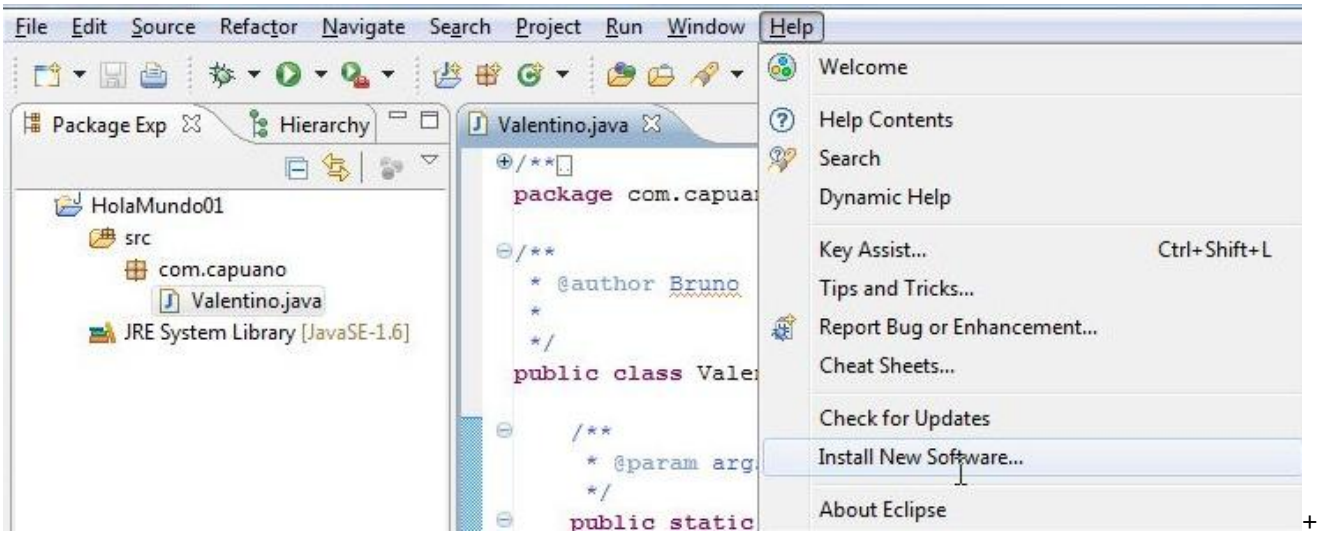
Como herramienta de gestión de versiones se utilizará TSF (Team Foundation Server). Eclipse por defecto no trae ningún plugin de conexión a este tipo de repositorio, por lo que el primer paso a realizar será la instalación de un plugin que permita trabajar desde Eclipse con el repositorio de versiones.

1.- Instalación del plugin de TFS2010 en Eclipse

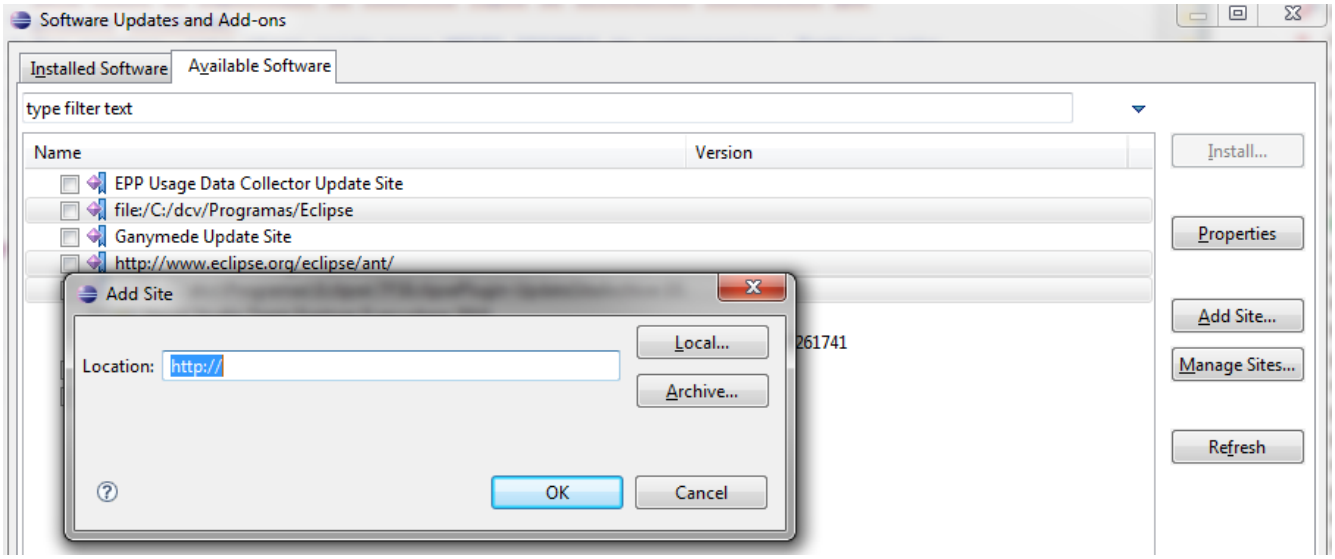
Lo primero que hay que hacer es descargarse el plugin de Eclipse para su conexión con el TFS2010, de la ruta:

http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=af1f5168-c0f7-47c6-be7a-2a83a6c02e57

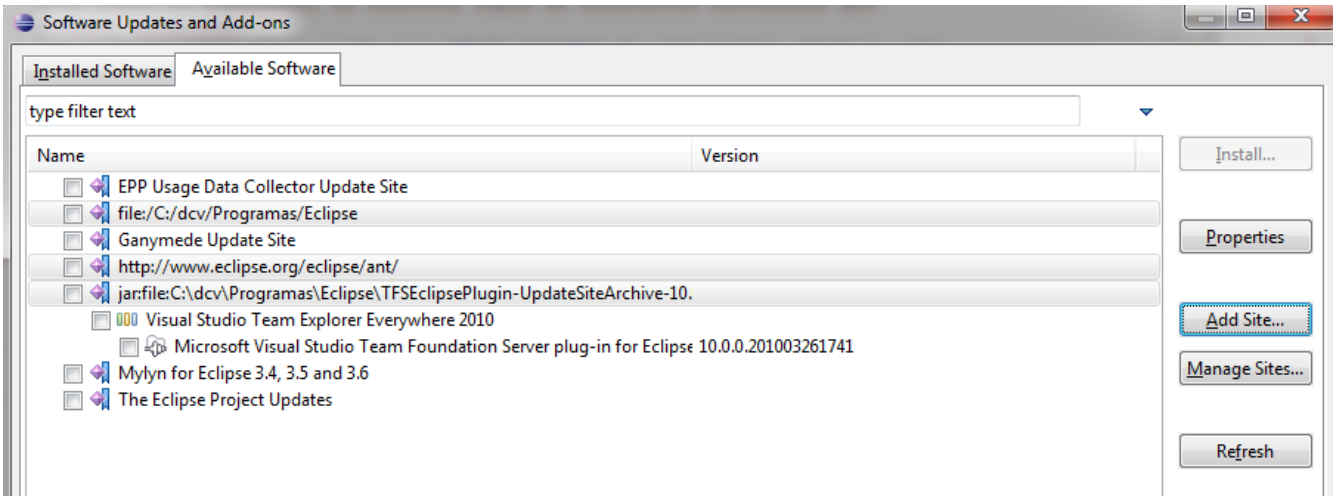
A continuación en la barra de opciones de Eclipse, se selecciona Help > "Install New Software".



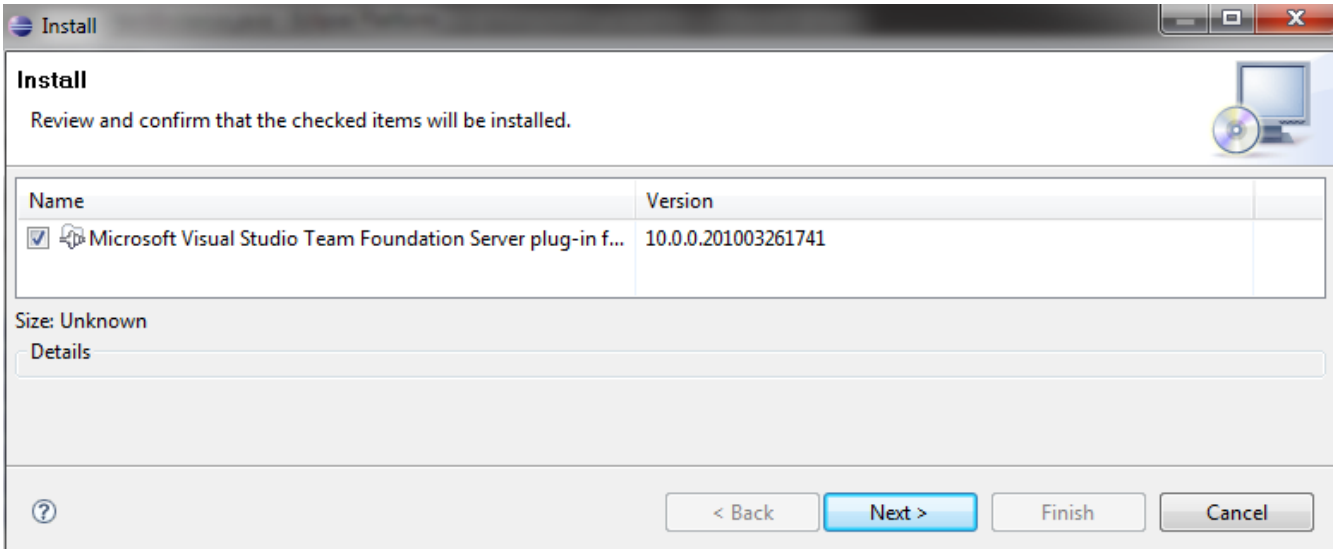
En la pestaña de “Available Software” se pulsa en “Add Sites”



Y después en “Archive”.
En la ventana emergente se selecciona el fichero de la instalación: “TFSEclipsePlugin-UpdateSiteArchive-10.0.0.zip”

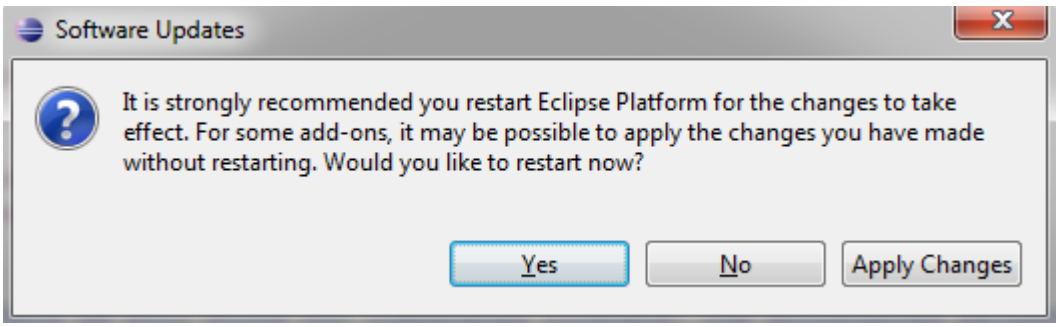


Se selecciona el plugin que se desea instlar “Visual Studio Team Explorer Everywhere 2010” y es pulsa “Install”

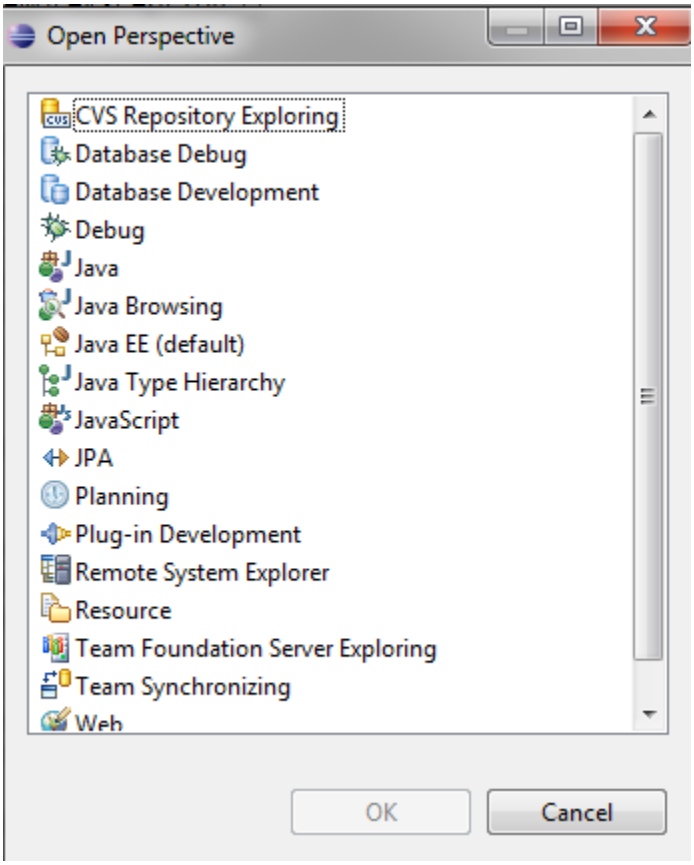


Se pulsa “Next”, se aceptan los términos de la licencia y a continuación se pulsa “Finish”

Cuando termine la instalación aparecerá un mensaje en el que recomienda el reinicio del programa. Se pulsa “Yes”



Una vez reiniciado Eclipse, en la ventana que muestra todas las vistas disponibles: “Window > Open perspective > Other” aparecerá la nueva vista del TFS: “Team Foundation Server Exploring”



El siguiente paso será configurar este plugin para realizar la conexión.

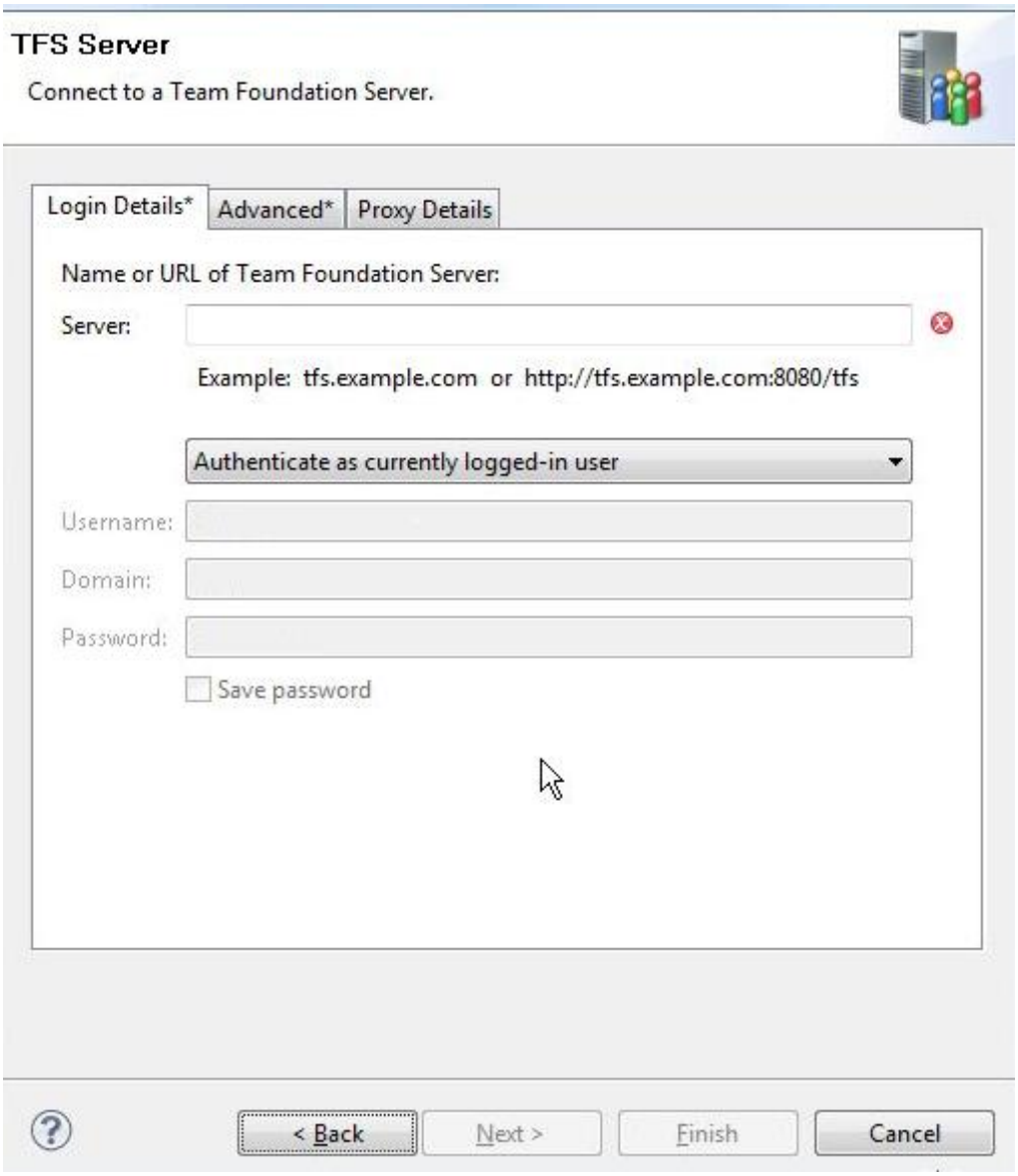
2.- Conexión plugin (Microsoft Visual Studio Team Explorer Everywhere) con TFS

A la hora de instalar el plugin, se puede indicar que se va a utilizar la versión trial (90 días) o si se dispone, introducir la licencia del mismo _

En el Eclipse, ir a File/Import, seleccionar Team/Team Foundation Server, se mostrará una pantalla donde se solicitará ingresar el número de licencia, la licencia a ingresar será la siguiente:

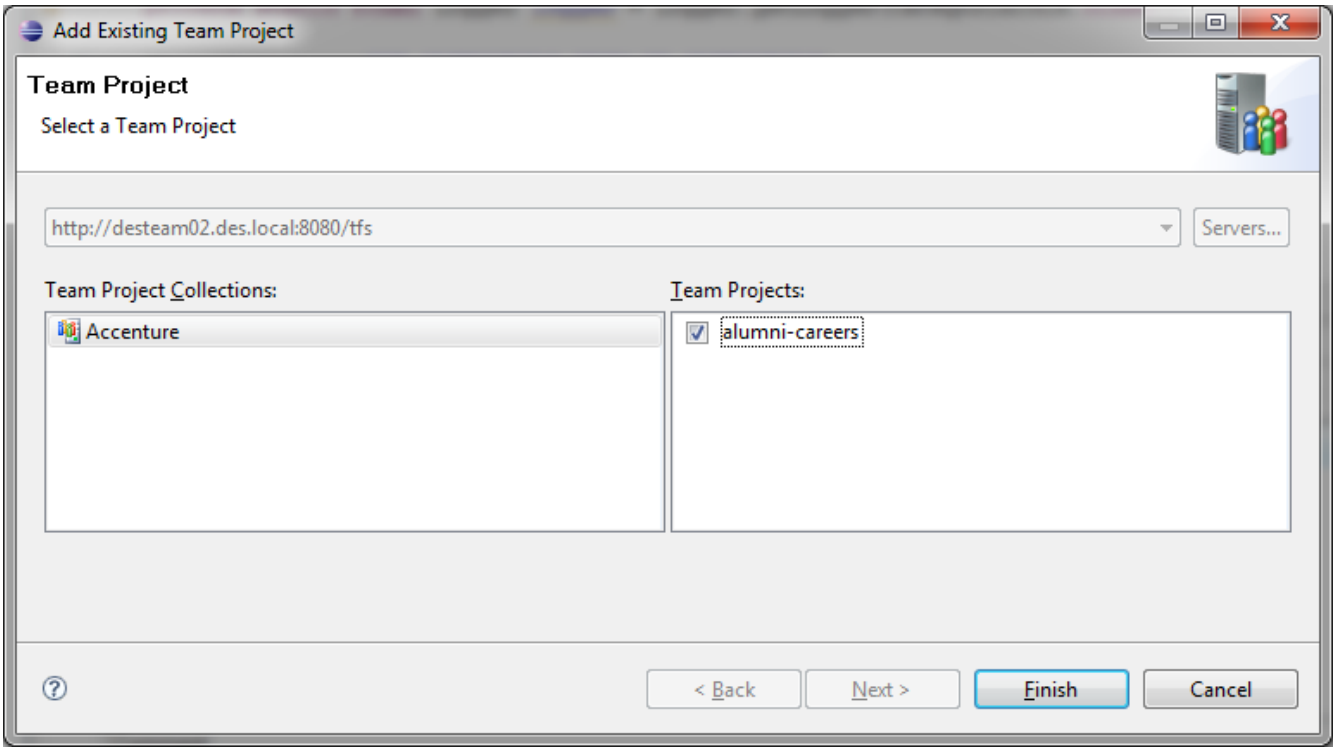
LICENCIA DEL TEAM EXPLORER				
7w3rj	**4wx3r**	**bv8jm**	**fc8p7**	**3w7qx**

En la siguiente ventana:

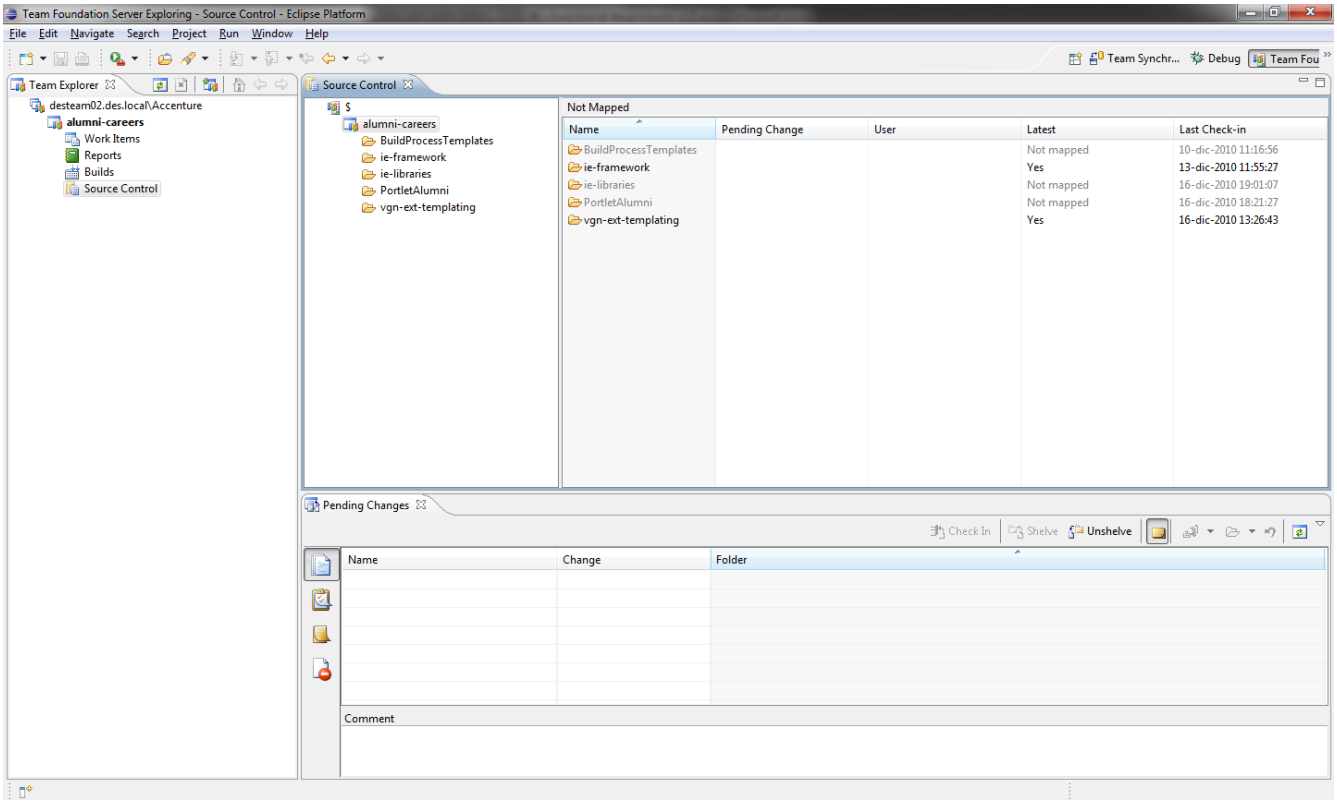


Será necesario indicar el servidor : http://desteam02.des.local:8080/tfs el usuario y password.

Luego se indica dentro de la colección Accenture, el proyecto “alumni-careers”, que es el último paso que hay que realizar para terminar la conexión.



Una vez se haya terminado el paso anterior, se podrá comenzar a trabajar con el gestor de versiones desde la herramienta de desarrollo.



3.3 Instalación y Configuración del Tomcat

Tomcat funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. La versión a utilizar será tomcat 6.0.x

La ruta donde descargar Tomcat 6.0.x es http://tomcat.apache.org/download-60.cgi

A continuación se procede a descomprimir el archivo descargado (en este caso, apache-tomcat-6.0.29-windows-x64.zip) en una ruta cualquiera (en este caso dentro de C:\Program Files).

Es necesario crear la variable de entorno CATALINA_HOME cuyo valor será para este caso C:\Program Files\apache-tomcat-6.0.29

Para arrancar el tomcat ejecutar el fichero startup.bat

3.3.1 Debug en Tomcat con Eclipse

Para realizar debug desde eclipse, realizar lo siguiente:

1. Arrancar Tomcat de modo debug

   Crear las siguientes variables de entorno:

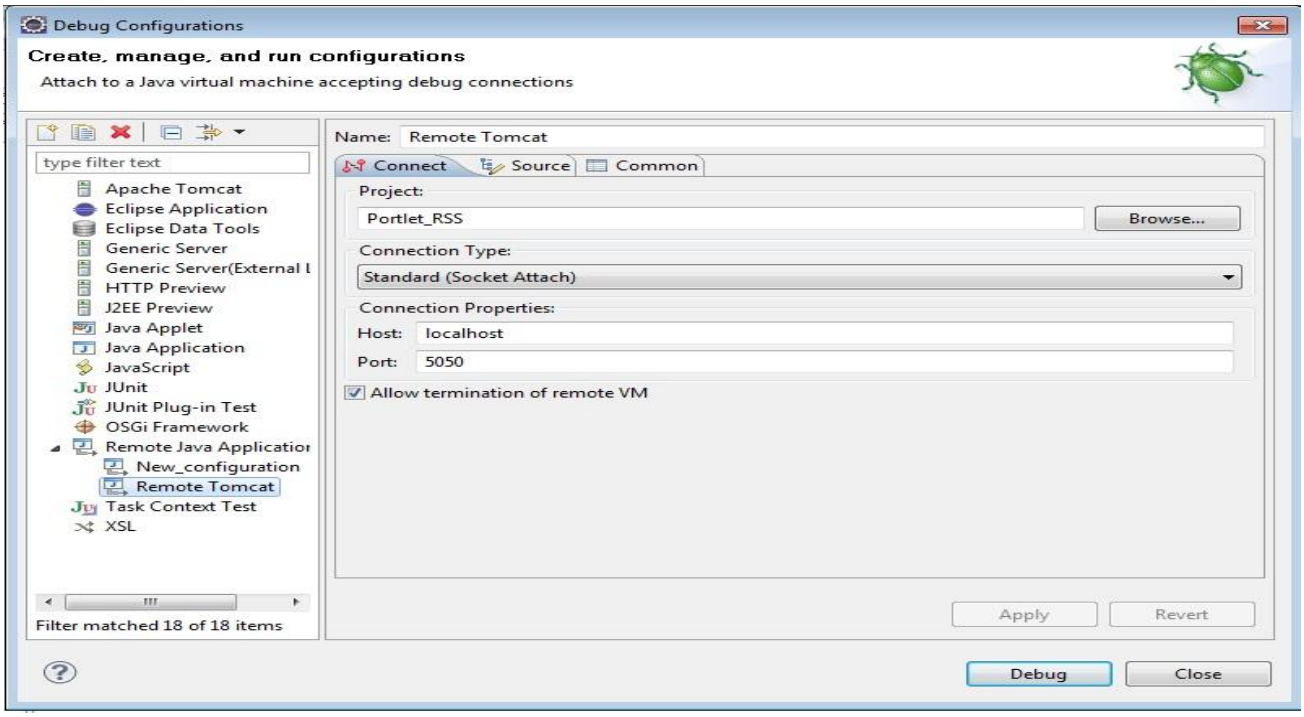
   ```
   JPDA_TRANSPORT=dt_socket
   JPDA_ADDRESS=5050
   ```

   Ubicarse en la carpeta bin del tomcat y ejecutar la siguiente instrucción desde un consola dos:

   ```
   catalina jpda run
   ```

2. Configurar una Aplicación Remota desde eclipse

   En Debug Configurations, crear y configurar un Remote Java Application, a continuación se muestra una interfaz como la siguiente:



Selecciona el proyecto a realizar debug e ingresa los siguientes datos:

Host: localhost

Port: 5050

Activa el check “Allow termination of remote VM” y presiona el botón Debug.

Ejecuta la aplicación, esta se detendrá en los puntos de interrupción que marquemos.

3.4 Instalación y Configuración de Ant

Apache Ant es una herramienta usada para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es similar a Make pero desarrollado en lenguaje Java y requiere la plataforma Java. La versión a utilizar será la última disponible.

La ruta donde descargar Apache Ant es: http://ant.apache.org/bindownload.cgi.

A continuación se procede a descomprimir el archivo descargado (en este caso, apache-ant-1.8.2-bin.zip) en una ruta cualquiera (en este caso dentro de C:\Program Files)

Es necesario crear la variable de entorno ANT_HOME cuyo valor será para este caso C:\Program Files\apache-ant-1.8.2

3.5 Instalación y Configuración de Contenedor de Portles

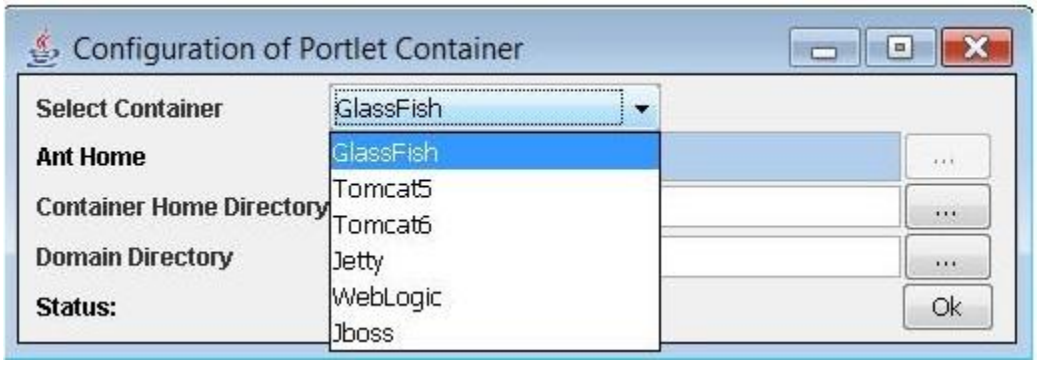
Portlet Driver permite construir rápido y fácil portles en java.

La ruta donde descargar Portlet Driver es: https://portlet-container.dev.java.net/public/Download.html

Descargar y seguir las siguientes instrucciones de instalación de portlet-driver para tomcat6:

1. Instalacion

   Ejecutar el siguiente comando (**java -jar portlet-container-configurator.jar**), donde mostrará la siguiente interfaz grafica:



   Ingresar los siguientes datos y presionar OK.

Select Container:	Tomcat6
Ant Home:	C:\Program Files\apache-ant-1.8.2
Container Home Directory:	C:\Program Files\apache-tomcat-6.0.29
Domain Directory:	C:\Program Files\apache-tomcat-6.0.29\webapps

2. Restaurar el tomcat
3. Revisar que el portlet-driver se encuentre desplegado (http://localhost:8080/manager/html/)
4. Probar el portlet-driver (http://localhost:8080/portletdriver/dt).
5. Realizar una copia de seguridad del war de portlet-driver (C:\Program Files\apache-tomcat-6.0.29\webapps\ portletdriver.war) en algún directorio local.

4 Entorno local

A continuación se procede a describir los diferentes proyectos que se deben tener montados en local para el desarrollo de un proyecto con Vignette.

En primer lugar habría que aclarar que el número de proyectos dependerá del tipo de desarrollo y que por lo tanto, no en todos los casos será necesario incluir algunos de estos proyectos.

En cualquier caso, se identifican cinco tipos de proyectos diferentes, que son los que se van a detallar en este capítulo: “ie-framework”, “ie-vgn-ext-templating”, “ie-<application>-vcm-extensions, ie-<application>-components y “ie-<porlet>”

Nota:
Se utilizará el literal: <application> para hacer referencia a cualquier aplicación o portal que se desea desarrollar, por ejemplo: “alumni”. Se utilizará el literal: <portlet> para indicar un portlet cualquiera, por ejemplo el portlet de login podría

- “**ie-framework**” es un proyecto Java (“Java Proyect” en Eclipse) que incluirá las clases comunes para todas las aplicaciones del Instituto de Empresa. Albergará por ejemplo beans y tipos de contenidos genéricos que puedan reutilizarse en una todos o casi todos los portales de la compañía, como noticias, artículos, …También incluyen carpetas con las clases específicas de cada aplicación.

- “**vgn-ext-templating**” será un proyecto web dinámico (“Dynamic Web Projecten Eclipse) que habrá que desarrollar a partir del “vgn-ext-templating” para incluir los jsp específicos de renderización de contenidos.

- “**ie-<application>-vcm-extension**” serán proyectos web dinámicos, que como su propio nombre indica, incluye las extensiones de VCM (widgets) cuando sea necesario. No todos los portales requieren extensiones de VCM.

- “**ie-components”**. Proyecto dinámico, como los dos anteriores, que contendrá los jsp de grids, sytles, secondary-pages,… La particularidad que presenta, es que los contenidos del mismo estarán también en Vignette, por lo que puede considerarse que los jsp que se encuentran en Eclipse son una copia de seguridad de los que están en el gestor de contenidos.

- “**ie-<portlet>**”. Son proyectos web dinámicos, con el código del portlet. Los desarrollos podrán tener entre 0 y n proyectos de este tipo, puesto algunos no utilizarán ningún portlet, otros dos, otros tres,

- “**ie-librerias-comunes**”. Este proyecto incluirá exclusivamente los jars comunes requeridos por los diferentes las aplicacines anteriores, no será por tanto, necesario definirlo como proyecto web dinámico. Como mínimo, contendrá los siguientes ficheros:
  thirdparty-combined.jar
  vgn-appsvcs-cda.jar y
  vgn-appsvcs-cma.jar

Que se encuentran en la ruta: /opt/Vignette/VCM/Content/8_0/lib/sdk

4.1 Estructura del proyecto: ie-framework

A continuación se detalla la estructura de paquetes Java que debe tener el proyecto ie-framework, y que debe incluir al menos los siguientes paquetes:

- **ie.cfg**
  En esta carpeta se guardarán todos los ficheros de configuración genéricos

- **ie.cfg.entorno**
  En esta carpeta se guardarán los ficheros de configuración genéricos para todo el Instituto de Empresa, dependientes del entorno. Se propone separar estos archivos dependientes del entorno para tenerlos identificados puesto que su despliegue no podrá ser automático y requerirá intervención manual para modificar su contenido en función de

- **ie.ctes**
  Contendrá todas las clases de constantes

- **ie.beans**
  Aquí se ubicarán las superclases de gestión de los beans que mapean el tipo de contenido.

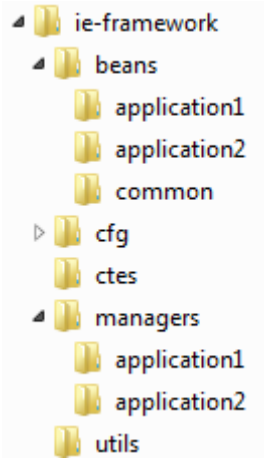
- **ie.beans.commons**
  Contendrá los diferentes tipos de contenidos comunes a todos los portales.

- **ie.beans.<application>**
  Contendrá los contenidos específicos a una aplicación o portal concreto, así para Alumni, se creará el paquete: ie.beans.alumni

- **ie.managers**
  Incluye las clases genéricas que dan soporte a los jsp para la recuperación de contenidos.

- **ie.managers.<application>**
  Incluye las clases específicas que dan soporte un jsp específico.

- **ie.utils**
  Paquete de utilidades

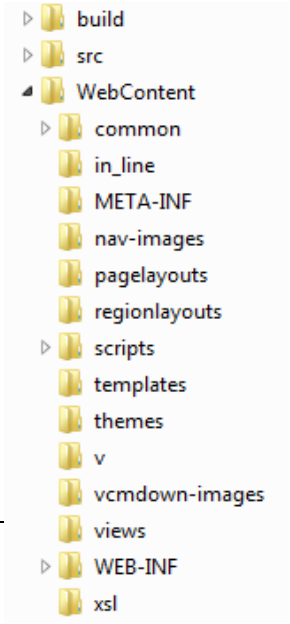


4.2 Estructura de los proyecto: vgn-ext-templating

Para la creación de este proyecto se seguirán las pautas indicadas en la documentación oficial proporcionada por Vignette en el documento:"Vignette Dynamic Portal Module and Vignette Dynamic Site Module 8.0 Developers Guide.pdf".

Brevemente, los pasos a seguir son:

1. Obtener la versión distribuida por Vignette del archivo: vgn-ext-templating.war ubicada en la ruta <directorioInstalacion>/Content/8_0/lib/sdk/
2. Descomprimir el war anterior en local
3. Borrar todos los ficheros que se encuentran en la carpeta "templates" del war



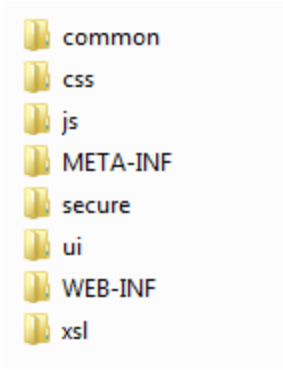
4. Depositar los jsp renderer de la aplicación de acuerdo a la siguiente estructura: una carpeta específica por aplicación y otra genérica para todos los elementos comunes a las diferentes aplicaciones.

Para un mayor detalle de cómo realizar las tareas anteriores, consultar la documentación oficial.

En resumen. La estructura del proyecto "vgn-ext-templating", viene ya establecida por Vignette y se respetará tal y como está definida, a excepción de la carpeta "template" comentada.

4.3 Estructura de los proyectos: ie-<application>-vcm-extensions

Para la creación de este proyecto, se seguirán las pautas indicadas en la documentación oficial proporcionada por Vignette en el documento: "Vignette Content Extensibility SDK Guide 8.0 SP1.pdf"



Los pasos a seguir son:

1. Obtener la versión distribuida por Vignette del archivo: vgn-appsvcs-cma.war ubicada en la ruta <directorioInstalacion>/Content/8_0/lib/sdk/
2. Descomprimir el war anterior en local
3. Incluir los jsp, imágenes,… manteniendo la estructura de directorios
4. Añadir las librerías: vgn-appsvcs-cma.jar, thirdparty-combined.jar y los jar específicos del widget en la ruta WEB-INF/lib

Para un mayor detalle de cómo realizar las tareas anteriores, consultar la documentación oficial.

4.4 Estructura de los proyectos: ie-components

Este proyecto estará estructurado a nivel de aplicación, y dentro de cada una de ellas, contendrá las siguientes carpetas o directorios:

- **<application>/jsp/portal/grid**
  Extensión de los ficheros ubicados en esta carpeta: jsp
  Fichero jsp generados por Vignette para establecer el grid de la página

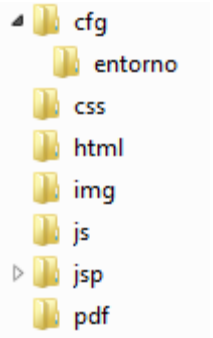
- **<application>/jsp/portal/styles**
  Extensión de los ficheros ubicados en esta carpeta: jsp
  Ficheros jsp generados por Vignette para establecer los estilos del portal.

- **<application>/jsp/portal/stylesTypes**
  Extensión de los ficheros ubicados en esta carpeta:jsp
  En esta ruta se almacenan los ficheros jsp de los diferentes tipos de estilos

- **<application>/jsp/portal/secondary-pages**
  Extensión de los ficheros ubicados en esta carpeta:jsp
  En esta ruta se almacenan los ficheros jsp de las secondary-page

4.5 Estructura de los proyectos: ie-<portlet>

Propuesta de estructura de directorios para los proyectos de tipo portlet, basada en la especificación de Sun.



- **/cfg**

Extensión de los ficheros ubicados en esta carpeta: properties o cfg
Ruta donde se almacenan los ficheros de configuración que no dependen del entorno. Estos ficheros se van a poder subir entre entornos con el procedimiento automático que se describirá posteriormente.

- **/cfg/entorno**
  Extensión de los ficheros ubicados en esta carpeta: properties o cfg
  Ruta donde se guardan el o los fichero/-s de configuración que depende del entorno. Según las necesidades del proyecto o aplicación, podrá haber uno o más ficheros en esta carpeta. El objetivo en cualquier caso es disponer del menor número de ellos, porque su deploy tendrá que ser manual y por lo tanto más costoso.

- **/css**
  Extensión de los ficheros ubicados en esta carpeta:css
  Ruta donde se guardan las hojas de estilos del portal

- **/html**
  Extensión de los ficheros ubicados en esta carpeta: html
  Ubicación de los ficheros html

- **/img**
  Extensión de los ficheros ubicados en esta carpeta:jpg,png, gif,…
  Ruta donde se almacenarán las imágenes que no sean gestionadas por Vignette

- **/js**
  Extensión de los ficheros ubicados en esta carpeta:js
  En esta carpeta se almacenan los ficheros java script

- **/jsp**
  Extensión de los ficheros ubicados en esta carpeta: jsp
  Resto de ficheros jsp. No se hará ninguna clasificación de los jsp en subcarpetas en función del canal porque es altamente probable reutilizar uno mismo jsp en varios canales.

- **/pdf**
  Extensión de los ficheros ubicados en esta carpeta: pdf
  Los documentos pdf se alojarán en esta carpeta

- **/WEB-INF**
  Extensión de los ficheros ubicados en esta carpeta: xml, cfg, properties
  Albergará los ficheros de configuración de la aplicación a nivel de servidor: web.xml, structs-config.xml,…

- **/WEB-INF/lib**
  Extensión de los ficheros ubicados en esta carpeta: jar
  Ruta donde se guardará los class y java de la aplicación empaquetados en un fichero de tipo jar.

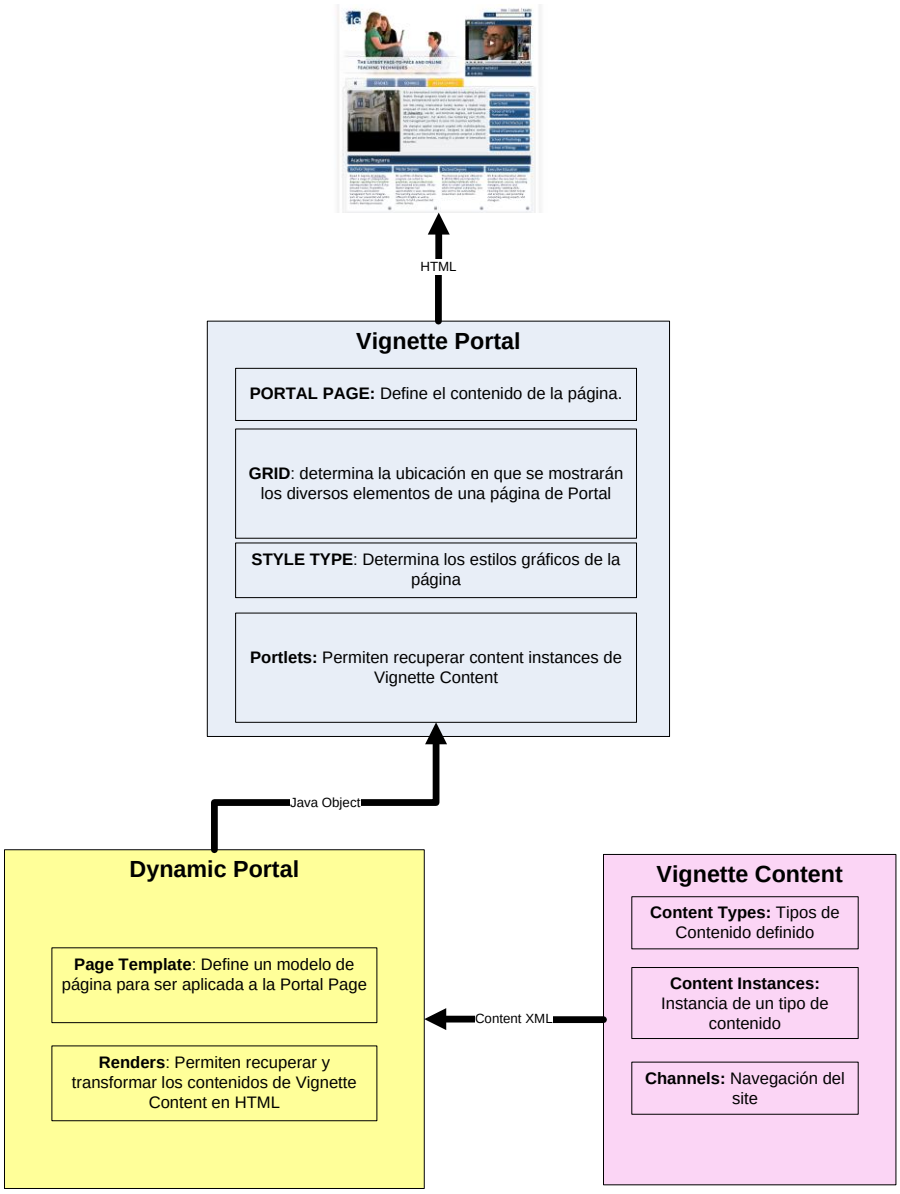
Una vez presentada la estructura de proyectos dentro de Eclipse, el siguiente capítulo va a abordar la otra rama fundamental para el desarrollo de portales, que es la propia herramienta de Vignettte, y más concretamente se van a explicar cómo se va a realizar la creación de contenidos, plantillas y portlets.

5 Desarrollos en Vignette: Introducción

Vignette Content suite se compone fundamentalmente de 3 aplicaciones para crear, gestionar y publicar un portal web.

Las aplicaciones que la componen son:

- **Vignette Portal** : Se encarga de la estructura, navegación y estilo de presentación de un site
- **Dynamic Portal** : Se encuentra muy relacionado con Vignette Portal porque premite recuperar y transformar los contenidos de Vignette Content en HTML
- **Vignette Content** : Se definen los tipos de contenido, instancias y canales del site



A continuación se detallan las rutas de logs para los diferentes módulos de Vignette desplegados en las máquinas de desarrollo

Entorno: Desarrollo

		Maquinas desarrollo		
		desvgwb01.des.local	**desvgman01.des.local**	**desvgdel01.des.local**
Módulos	**DPM**		/data/logs/DPM	/data/logs/DPM
	Portal		/data/logs/Portal	/data/logs/Portal
	VCA		/data/logs/VCA	/data/logs/VCA
	VCM	/data/logs/VCM	/data/logs/VCM	/data/logs/VCM
	VCS		/data/logs/VCS	/data/logs/VCS
	Collab		/data/logs/Collab	/data/logs/Collab

Módulo DPM: Logs de Dynamic Portal.

Módulo Portal: Logs de Portal.

Módulo VCM: Logs de Vignette Content Manager.

Módulo VCA: Logs de Vignette Community Application.

Módulo VCS: Logs de Vignette Collaboration Services.

Módulo Collab: Logs de Vignette Collaboration.

En ésta guía se explican los desarrollos más habituales en la suite de Vignette:

- Vignette Portal: Grids, StylesTypes, Portal Pages
- Dynamic Portal: Renders, Display Views
- Vignette Content: Content Types, Content Instances

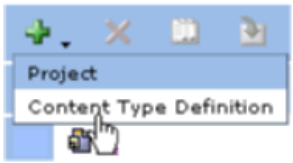
6 Desarrollos en Vignette: Creación de tipos de contenido

Acceder a AppConsole de VCM.

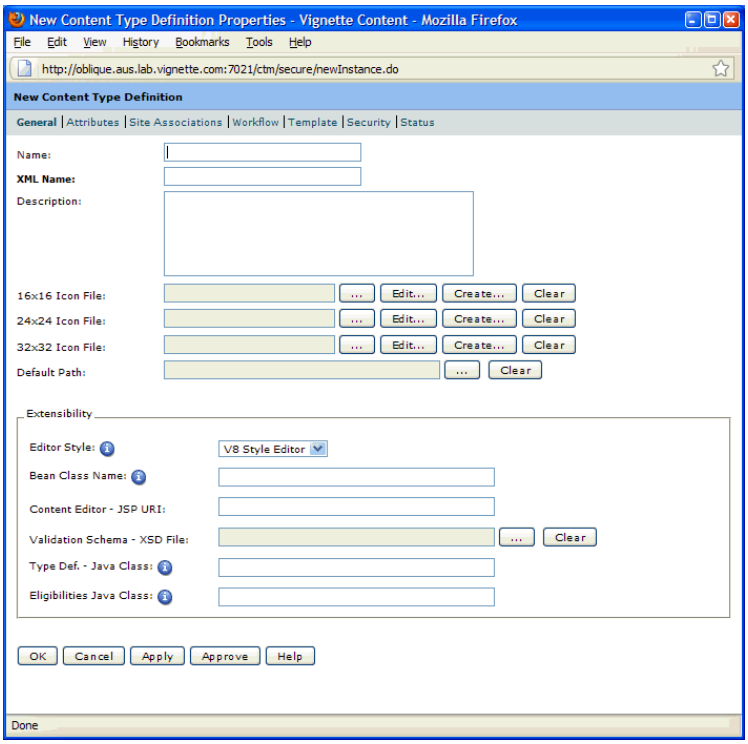
Navegar a la pestaña Workbench → Content Type Definitions.

Navegar (o crear) un proyecto para almacenar el tipo de contenido a crear.

Seleccionar la opción “Content Type Definition” del menú New.



Se presenta la **pestaña de propiedades generales** del nuevo tipo de contenido.

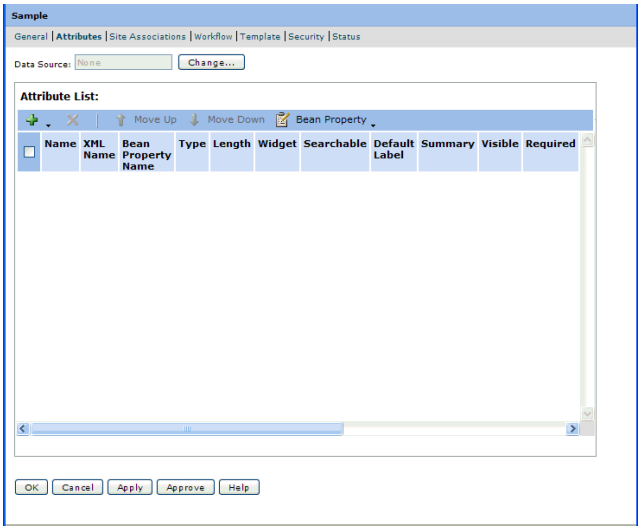


Dar de alta los atributos especificados en el documento técnico correspondientes a esta pestaña.

Field	Description
Name	Nombre del tipo de contenido.
XML Name	Nombre XML del tipo de contenido. **NOTE:** Debe ser único, sin espacios.
Description	Descripción del tipo de contenido.
Editor Style	Seleccionar **V8 Style Editor**.
Content Editor—	Introducir `/vgn-wcm-up/cif/defaultcif.jsp`

JSP URI	
Type Def—Java Class	Dejar en blanco hasta haber desarrollado Bean.

Pestaña de atributos:



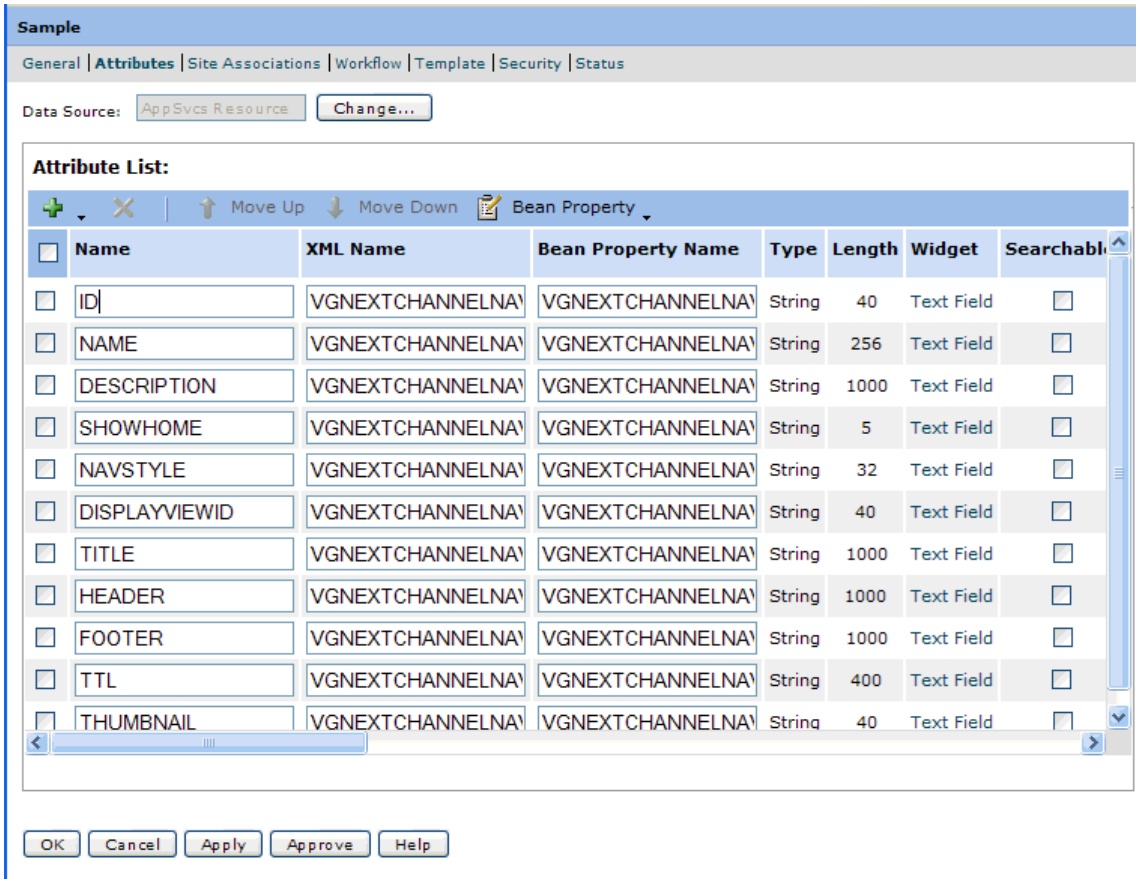
Se define el Data Source de donde se obtendrán los campos de las tablas a mapear como atributos del tipo de contenido y los widgets asociados a los tipos de contenido.

Para seleccionar el DataSource:

1. Pulsar el botón “Change” correspondiente al diálogo “Data Source”.
2. Se abrirá un selector con los orígenes de datos disponibles. Seleccionar el indicado en el diseño técnico.
3. Pulsar el botón de “Text Connection” para confirmar la conexión con la base de datos.
4. Guardar los cambios pulsando el botón “OK”.

Para importar atributos desde la base de datos:

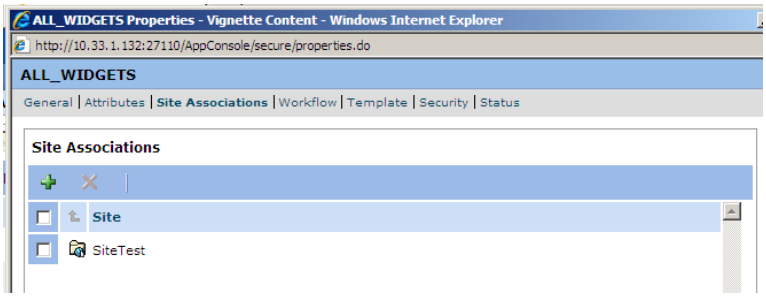
1. En la pestaña de Atributos, pulsar el botón + y seleccionar “Data Source Attribute”.
2. Seleccionar la tabla de la que se quieren importar los atributos.
3. Seleccionar los atributos a importar indicados en el documento técnico.



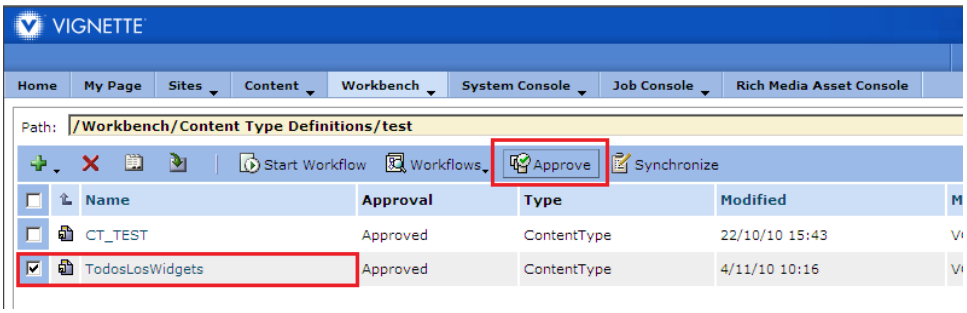
4. Rellenar las columnas ID, XML Name, Bean Property Name, Widget y marcar los checks indicados en el diseño técnico.

   Por defecto, se asocian los widgets Text Field a todos los campos del tipo. Hay que modificar estos widgets con los indicados en el diseño técnico, y configurar-los también según el diseño técnico.

5. Ir a la pestaña “Site Association” y asociar el widget creado a nuestro site.



6. Aprobar los cambios realizados en el widget.



6.1 Implementación de bean asociado al tipo de contenido

Entendemos por *bean* toda clase que contenga una serie de propiedades accesibles para lectura y, opcionalmente, escritura mediante métodos conocidos por *getters* y *setters*. Se han de cumplir las normas de codificación especificadas en http://java.sun.com/docs/codeconv/CodeConventions.pdf.

En la mayoría de casos estos *beans* estarán asociados a tipos de contenido, para tener mapeados los atributos de una instancia de contenido concreta a cada una de las propiedades del *bean*.

En este caso, todos los beans desarrollados deben de extender de la clase **`ie.bean.ContentInstanceBaseBean`**.

```
package ie.bean.<identificador portal>;

public class WebComodinBean extends ContentInstanceBaseBean {…
```

Esta clase pone a disposición de sus clases hijas un objeto org.apache.log4j.Logger y que se debería de redefinir como una variable estática:

```
private static final Logger logger = Logger.getLogger(Categoria.class);
```

La clase deberá ir precedida de la anotación `@ContentType(name="<nombre tipo contenido Vignette>")`.

El siguiente paso es definir las propiedades del bean, que normalmente se corresponderá con la totalidad de atributos de la instancia de contenido que representa y se crean los respectivos métodos *getter* y *setter* para cada una de ellas.

```
private String nombre = "";
private Integer tipo = null;
public String getNombre() {
return nombre;
}
public String setNombre(String nombre) {
this.nombre = nombre;
}
public Integer getTipo() {
return tipo;
}
public void setTipo(Integer tipo) {
```

```
this.tipo = tipo;
}
```

A partir de aquí, se han de establecer las relaciones de atributos, relaciones, content selects y static files de las instancias de contenido con los campos de la clase.

Para ello se deben utilizar una serie de anotaciones definidas a tal efecto:

- Para designar un atributo, el campo que lo ha de recoger ha de ir precedido por la anotación `@Attribute`, que recibe un único parámetro "`name`" donde se indica el nombre XML del atributo:

```
@Attribute(name="IE-CATEGORIA-ID-CATEGORIA")
private String nombre = null;
```

name: Nombre XML del atributo. (ver Ilustración. Obtener name attribute annotation).

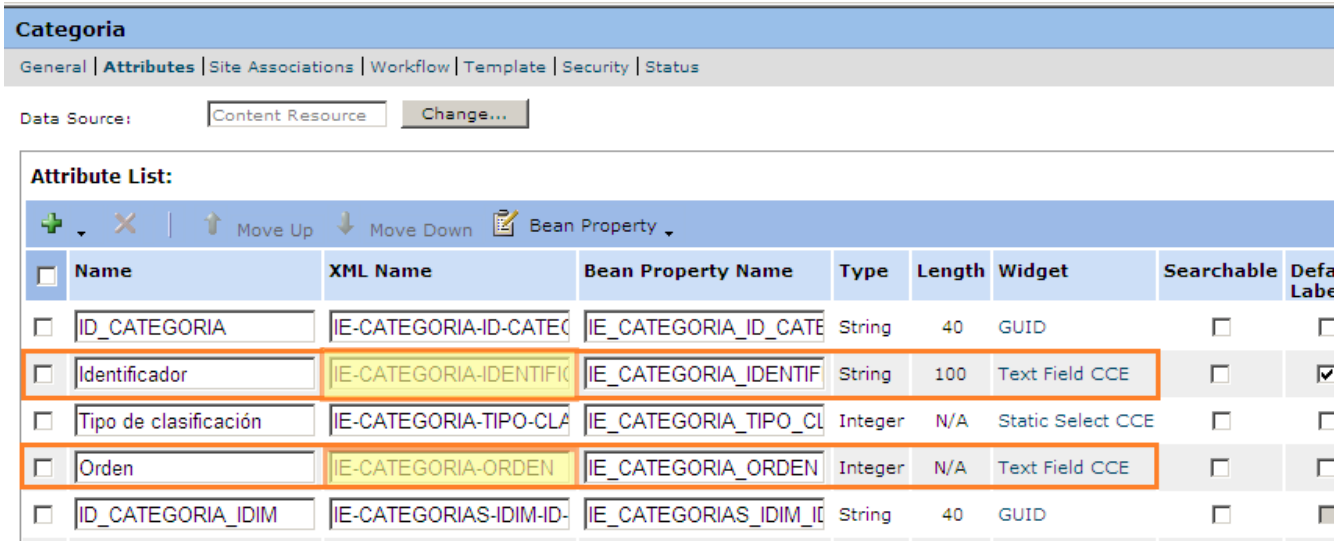


Ilustración 1 Obtener name attribute annotation

- Para designar un *content select*, el campo que lo ha de recoger (que ha de ser una implementación de `ContentInstanceBaseBean`) ha de ir precedido por la anotación `@ContentSelect`, que recibe un único parámetro "`name`" donde se indica el nombre del atributo:

```
@ContentSelect(name="IE-CATEGORIA-ID-CONTENT-SELECT")
private WebTestImageBean imagen = null;
```

name: Nombre XML del atributo. (Igual que para un Atributo, el campo Name XML del atributo).

- Para designar un *static file*, el campo que lo ha de recoger (que ha de ser `java.lang.String`) ha de ir precedido por la anotación `@StaticFile`, que recibe un único parámetro "`name`" donde se indica el nombre del atributo:

```
@ContentSelect(name="IE-CATEGORIAS-ID-STATIC-FILE")
private String pathCSS = "";
```

name: Nombre XML del atributo. (Igual que para un Atributo, el campo Name XML del atributo).

- Para designar un relator, el campo que lo ha de recoger (que ha de ser una implementación de `java.util.Map`) ha de ir precedido por la anotación `@Relator` o bien la anotación `@MultiLanguage` (dependiendo si la información contenida en el relator es dependiente del idioma), que recibe un parámetro "`name`" donde se indica el nombre de la relación (sin el prefijo, puesto que este se establece en el parámetro de aplicación `relator.prefix`), un parámetro "`type`" que indica la clase que contendrá cada elemento del relator y un parámetro "`key`" que indica el campo que se utilizará como clave dentro de la `java.util.Map`:

```
      @MultiLanguage(tablaRel="NOMBRE-TABLA-MULTIIDIOMA",
type=TestIdimBean.class, key="WEB-TEST-IDIM-ID-IDIOMA)
      private Map datosIdioma = null;
```

Key: Campo XML Name del atributo que se usará para generar las claves del mapa de clases relacionadas. En el caso de datos mulitidioma, éste será el idioma. (ver Ilustración 1. Obtener key annotation)

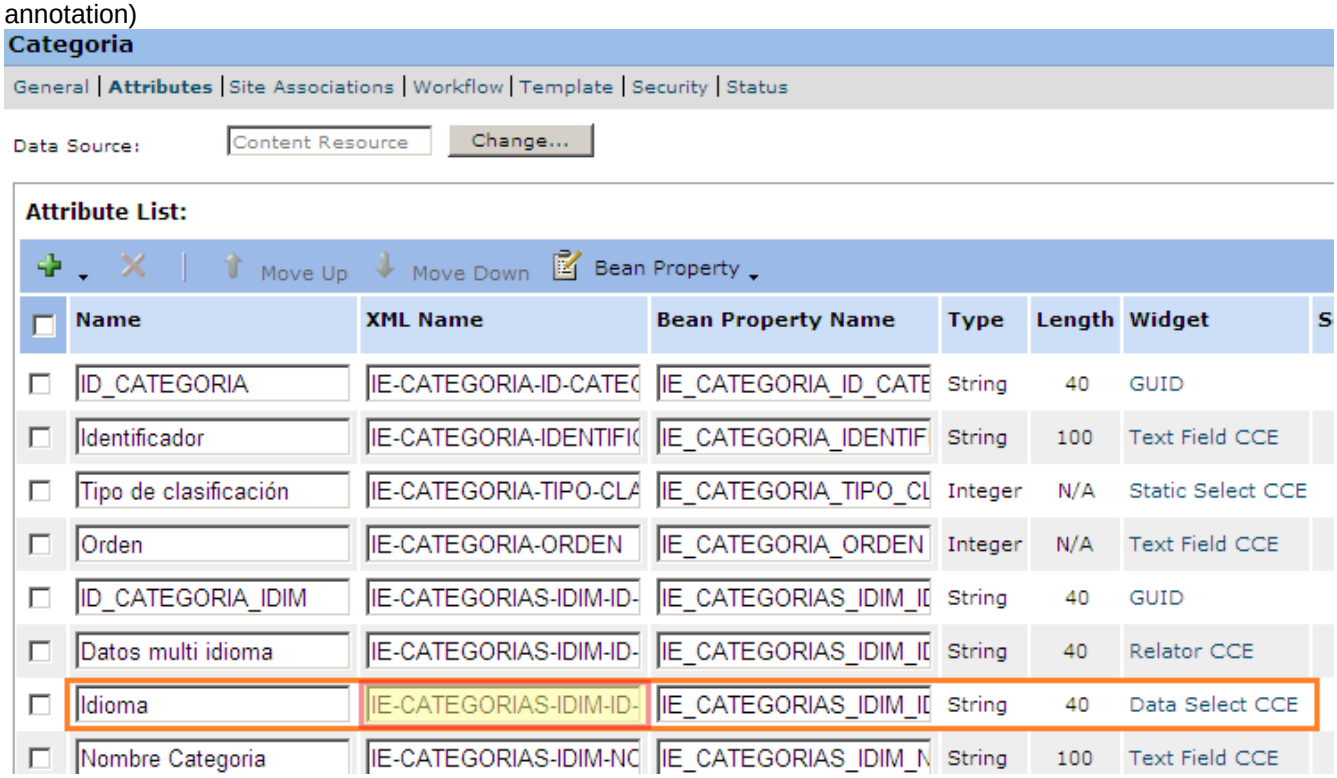


Ilustración 2 Obtener key annotation

Type: Nombre de la clase que contendrá la información del RelatorCCE definido.

tablaRel: Nombre de la tabla que contiene los datos del RelatorCCE. Esta tabla se puede consultar en la columna Relations del atributo que tiene definido el relator PERO sin el prefijo. (ver Ilustración 2. Obtener tablaRel annotation)

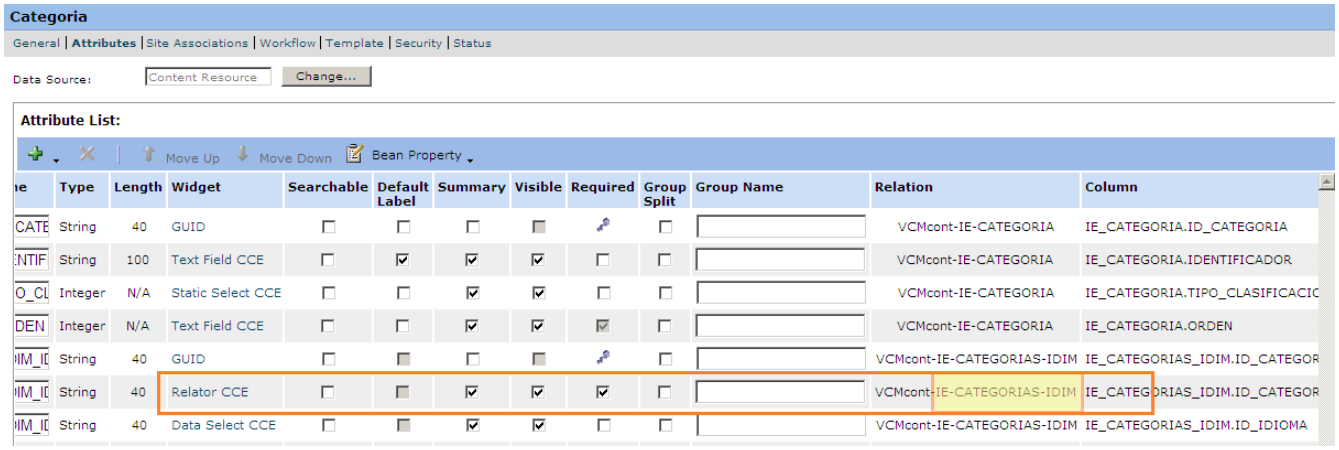


Ilustración 3 Obtener tablaRel annotation

Adicionalmente, la superclase también proporciona una propiedad, `selectedLanguage`, para establecer el idioma por defecto para los datos multiidioma.

Esto puede ser útil para crear métodos que encapsulen propiedades contenidas en un relator que sean dependientes del idioma.

Sobre el ejemplo anterior, si la clase tiene un Map que contiene instancias de `TestIdimBean`, se pueden escribir métodos para acceder a sus propiedades de forma directa.

En lugar de tener que escribir:

```
TestIdimBean idioma = (TestIdimBean)
test.getDatosIdioma(test.getSelectedLanguage());
String titulo = “”;
If(idioma!=null) {
      idioma.getTitulo();
}
```

Se puede definir un método getTitulo() dentro de la propia clase Categoria, que devuelva el título multi idioma asociado a la categoría, teniendo en cuenta el selectedLanguage definido previamente en la instancia:

```
public String getTitulo() {
if(datosIdioma == null || ¡datosIdioma.containsKey(selectedLanguage)) {
      return “”;
} else {
      return ((TestIdimBean) datosIdioma.get(selectedLanguage)).getTitulo();
}
}
```

Con lo que luego, desde una JSP u otra clase se podría obtener el título de la siguiente forma:

```
String titulo = test.getTitulo();
```

6.2 Nomenclatura Tipos de Contenido y Beans Asociados

En este apartado se contempla la definición de un tipo de contenido y el mapeo de este con una bean.

Tipo de contenido del VCM	IE-*<nombre_tipo_contenido>*
Bean asociado al tipo de contenido	*<Nombre_tipo_contenido>*

En caso que el tipo de contenido contenga atributos multiidioma, se debe crear una nueva clase que contendrá los atributos multiidioma.

Bean multiidioma	*<Nombre_tipo_contenido>*Idim

Ejemplo: Un tipo de contenido que represente un elemento comodín del portal tendrá:

Nombre del tipo de contenido: IE-Categoria

Atributos: Identificador, datos_multiidioma.

Datos multiidioma: Relator con idioma, título, descripción.

Nombre del bean asociado: Categoria

Atributos: String identificador, Map<String, CategoriaIdim> datos_multiidioma.

Nombre de la bean con los atributos multiidioma del tipo de contenido: CategoriaIdim.

Atributos: String idioma, String titulo, String contenido

7 Desarrollos en Vignette: Creación de plantillas

7.1 Descripción y utilización de Dynamic Portal

- Dynamic Portal se sitúa sobre la Gestión de Presentación para facilitar la entrega del contenido dinámico en tiempo real
- Permite el contenido dirigido, personalización y seguridad
- Genera y gestiona caché para el contenido dinámico
- El Workspace de presentación es la interfaz a través del cual los objetos de presentación son creados y gestionados
- El entorno de Preview muestra páginas renderizadas , permite editar contenido s en el contexto de la página
- Dynamic Portal utiliza las capacidades de los stages de Mangement y Delivery de Vignette.
- Dynamic Portal utiliza Vignette Portal para la creación y gestión de page templates.

7.2 Conceptos básicos

- ***Grid:*** se trata de un componente de *Vignette Portal* que determina la ubicación en que se mostrarán los diversos elementos de una página de *Portal,* tales como la cabecera, pies de página, menú de navegación (todos estos serán *Styles*) o la región donde se mostrará el contenido principal de la página.

  La apariencia de la *grid* viene determinada por un fichero JSP.

- ***Style y style type:*** un *estilo* de VAP es un componente que controla la apariencia de una región determinada de una página de VAP. Un *tipo de estilo* es una abstracción que corresponde a una determinada región de una página y de la que los *estilos* son instancias concretas. Algunos ejemplos de tipos de estilo podrían ser *cabecera*, *pie de página, menú de navegación vertical, menú de navegación horizontal,* ect. Ejemplos de *estilos* serían *pies de página, cabeceras* o *menús de navegación* concretos.

  La localización de cada estilo dentro de una página viene controlada por la *grid.*

  Para cada *tipo de estilo* se puede seleccionar un *estilo* concreto a usar en cada página. Por otro lado se pueden usar estilos por defecto para todo el Portal, cuando un estilo concreto no haya sido seleccionado para una determinada página.

  Los *tipos de estilo* vienen a estar definidos por dos ficheros:

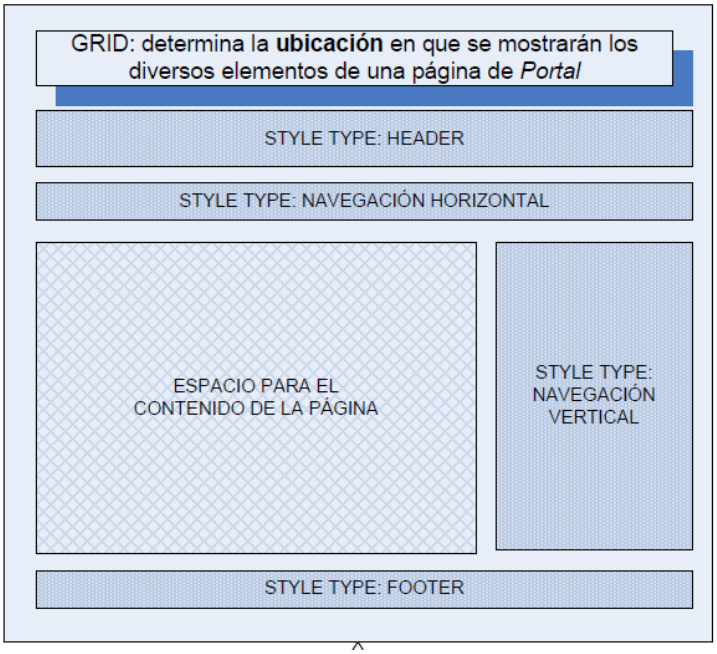
  - *template_header.inc*: proporciona código que todos los *estilos* de un tipo de estilo concreto pueden usar. Variables y funciones declaradas en este fichero no deben ser declaradas en la JSP primaria y otros ficheros asociados a los *estilos* pertenecientes al *tipo de estilo.*
  - *component.xml*: necesario para desplegar el *tipo de estilo* empleando el sistema de despliegue.

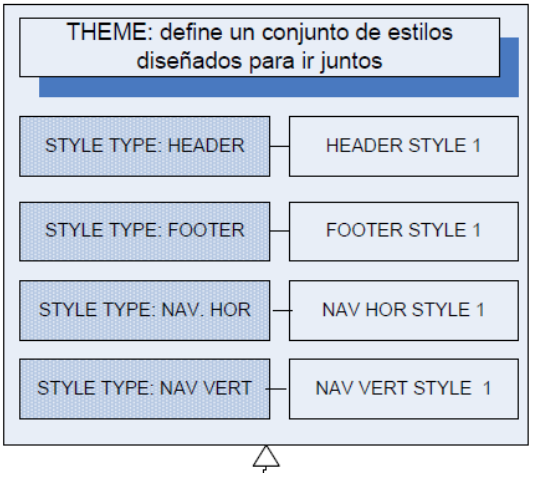
  Los estilos se definen mediante dos ficheros:

  - JSP primaria: contiene el código necesario para ejecutar el *estilo* que no esté compartido con otros *estilos*. Cuando este fichero es subido, *Portal* comprueba si para el *estilo* asociado hay que emplear un fichero de cabecera (*template_header.inc*). De ser así incluye el contenido de este en la JSp primaria.
  - Ficheros secundarios: imágenes, CSS, JavaScript, etc…

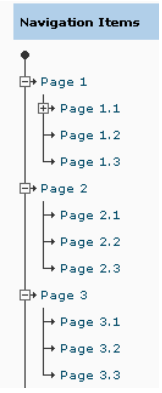
- *component.xml*: necesario para desplegar el *estilo* empleando el sistema de despliegue. Entre otros valores, en este fichero se define la propiedad *apply-template-header* que define si el estilo va a usar el *template_header.inc* del tipo de estilo asociado.

- ***Theme:*** se trata de un conjunto de estilos diseñados para ir juntos. A un determinado Tema se le pueden asignar una serie de estilos diseñados para ir juntos. Un tema puede ser asignado a nivel de ítem de menú, de site o de servidor***.***





- ***Portal Page***: Un Portal Page define el "Espacio para el contenido de la página" definido en la Grid. Para asociar una Grid con una Portal Page debemos crear un Navigation ítem de Portal.



- ***Navigation Item:*** *Vignette Portal* usa, para cada site, un *árbol de navegación* para representar la jerarquía de menús y submenús de dicho site. Dicho *árbol de navegación* está compuesto por componentes de *Portal* denominados *navigation items.* Los *navigation items* pueden ser creados a nivel raíz dentro del árbol o bien anidados unos dentro de otros hasta el nivel que sea necesario.

  A fin de controlar la apariencia de los *navigation items* existen los *navigation styles* y *navigation style types.* Éstos trabajan en conjunción con la *grid*, que controla la ubicación del de los *navigation ítems* dentro de la página que el usuario final ve.

- ***Page Template***: Permite crear y gestionar múltiples páginas del site con la misma estructura. Controla el *layout*, tema, regiones y componentes.

  Queremos presentar como páginas web: canales y contenidos definidos en VCM. Para ello, vamos a asociar los canales y contenidos a *page templates*.

  - Al asociar un canal a una *page template*, obtenemos un *channel page*.
  - Al asociar un contenido a una *page template*, obtenemos un *ítem page*.

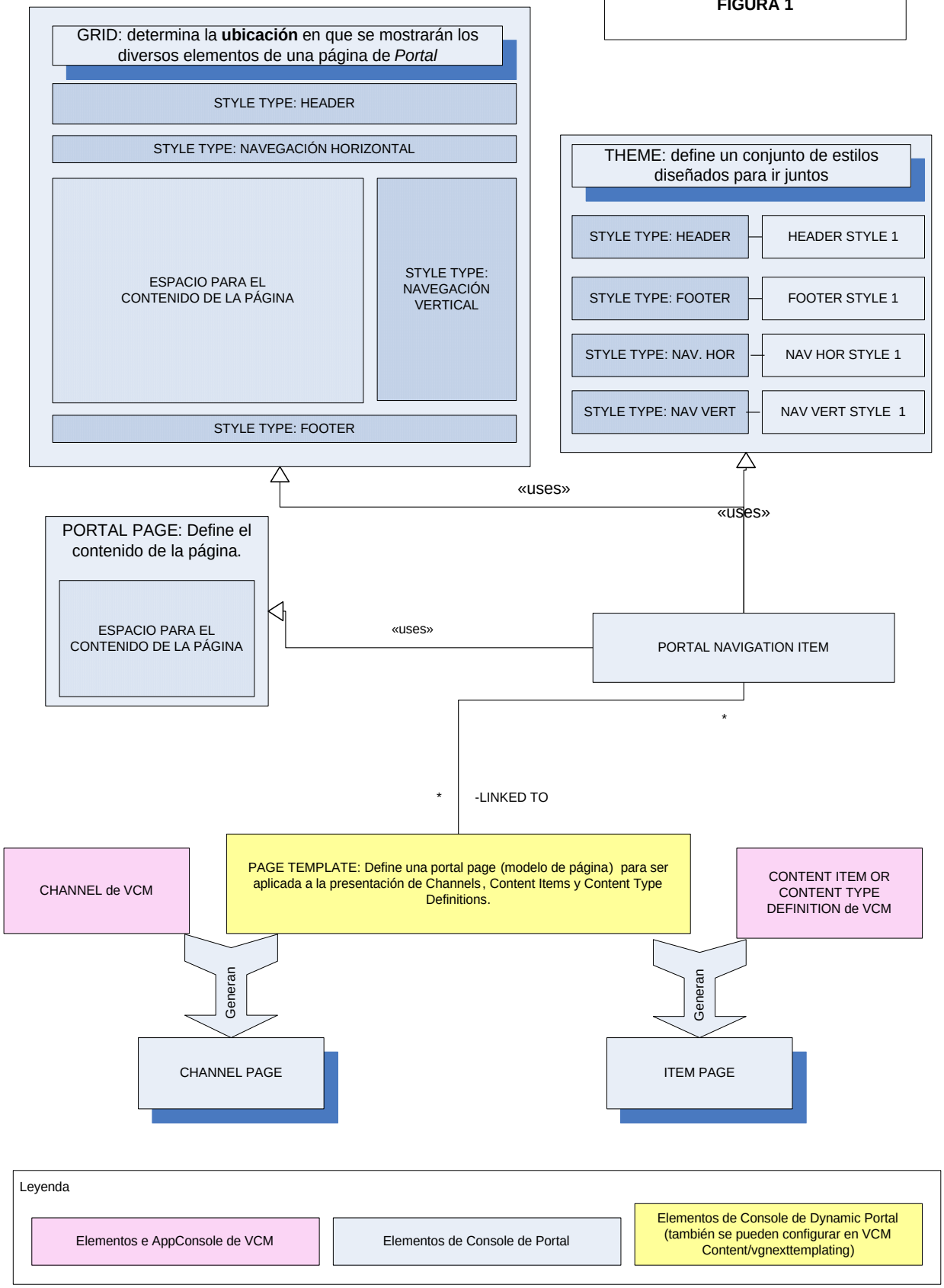
También se permite asociar Content Type Definitions a Page Templates para que todas las instancias de un tipo de contenido concreto se visualicen con la misma plantilla.

Hay 3 tipos de page templates:

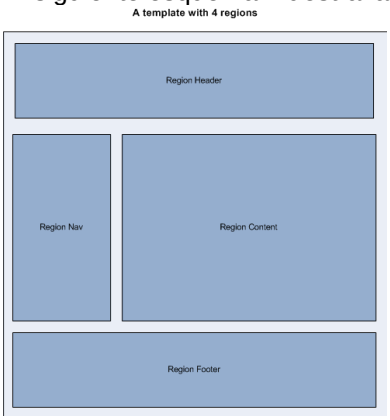
- JSP Page Template y Managed Page Template que aplican a Dynamic Site.
- Portal Page Template, que aplica a Dynamic Portal, y al desarrollo de nuestros portales.

A continuación se muestra un esquema que relaciona los conceptos expuestos. Este esquema representa la estructura general de una página del site (page layout).

FIGURA 1


GRID: determina la **ubicación** en que se mostrarán los diversos elementos de una página de *Portal*
STYLE TYPE: HEADER
STYLE TYPE: NAVEGACIÓN HORIZONTAL
ESPACIO PARA EL CONTENIDO DE LA PÁGINA
STYLE TYPE: NAVEGACIÓN VERTICAL
STYLE TYPE: FOOTER
THEME: define un conjunto de estilos diseñados para ir juntos
STYLE TYPE: HEADER
HEADER STYLE 1
STYLE TYPE: FOOTER
FOOTER STYLE 1
STYLE TYPE: NAV. HOR
NAV HOR STYLE 1
STYLE TYPE: NAV VERT
NAV VERT STYLE 1
«uses»
«uses»
PORTAL PAGE: Define el contenido de la página.
ESPACIO PARA EL CONTENIDO DE LA PÁGINA
«uses»
PORTAL NAVIGATION ITEM
*
*
-LINKED TO
CHANNEL de VCM
PAGE TEMPLATE: Define una portal page (modelo de página) para ser aplicada a la presentación de Channels, Content Items y Content Type Definitions.
CONTENT ITEM OR CONTENT TYPE DEFINITION de VCM
Generan
Generan
CHANNEL PAGE
ITEM PAGE
Leyenda
Elementos e AppConsole de VCM
Elementos de Console de Portal
Elementos de Console de Dynamic Portal (también se pueden configurar en VCM Content/vgnexttemplating)

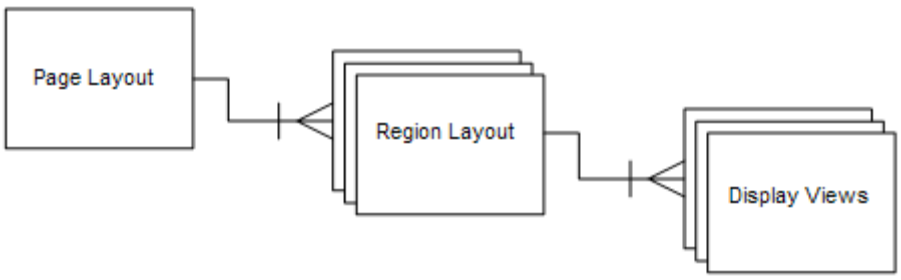
- ***Region***: Area de la página donde se muestra un contenido. Una *page template* se organiza mediante regiones. Mediante el bloqueo y desbloqueo de las regiones, se controla si la región podrá ser editada por el creador de la página o si la región estará fija para todas las páginas de la template.
  El siguiente esquema muestra la estructura de la página en regiones.



  Las regiones tienen un scope asociado:
  - Template Scope: Se configurará la región a nivel de template, y para todas las páginas que usen la template correspondiente se mostrará la misma configuración de Display View.
  - Page Scope: Se configurará la región a nivel de página. Cada página que haga referencia a la región deberá configurar las Display Views de dicha región.
- **Content Region:** Cada región puede contener uno o más componentes, y cada uno de estos componentes muestra diferentes tipos de contenido de Vignette Content Manager.
- **Componente**: Cada región tiene asociados uno o más componentes. Cada componente renderiza un tipo de elemento del VCM.

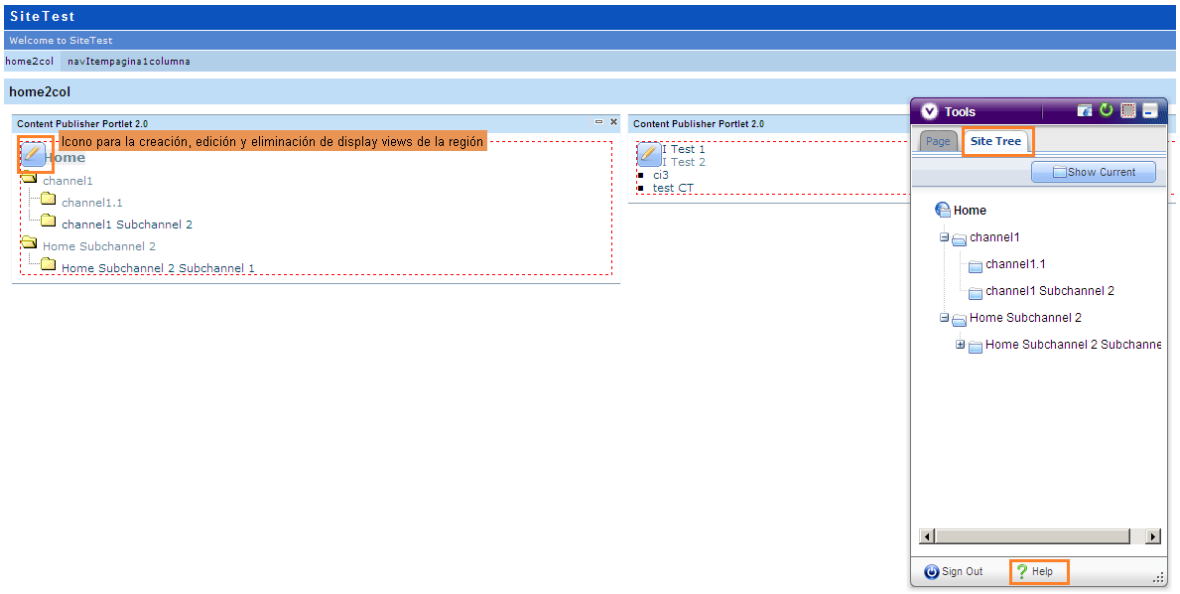
- **Render:** Los *components*, *regions* y *pages* generan código HTML especificado en los renders asociados. Se implementarán renders customizados que permiten controlar cómo el contenido se mostrará utilizando JSP. Permite acceder al contenido de VCM y transformarlo en HTML.
  Existen tipos de renders que van definiendo la estructura de las páginas.
  Los renderes, page layout, region layout y display view forman una página.
  Una página invoca al Page Layout, que a su vez invoca a los region layouts y a su vez invocan a los display views.

  La figura muestra el diagrama de una página



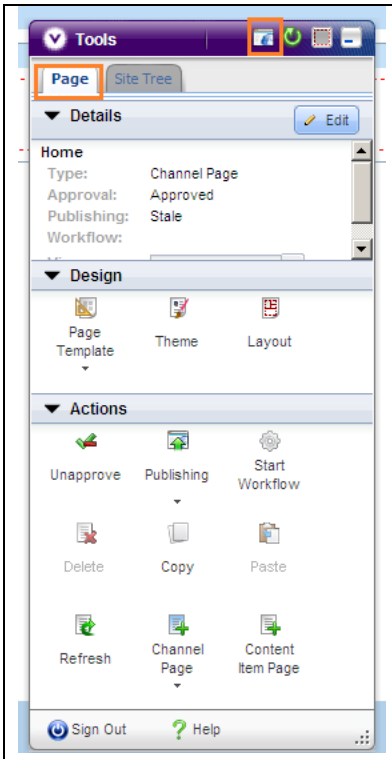
  Una *Region Layout* se corresponde con una definición de un estilo (ver punto Definir Estilos).
  Una Page Layout se corresponde con la definición de la grid y del contenido de la página.
  Una *Display View* define la visualización de un componente de dentro de una región.

- **Presentation workspace:** Permite configurar las regions de las páginas. Al acceder al site en modo edición se muestra una paleta que permite configurar los elementos de la página. Para conocer todas las opciones de la paleta de configuración se proporciona una URL de Ayuda.



Se presenta la paleta con la pestaña Site Tree que muestra el árbol de navegación del Site por canales.

El icono de Help redirecciona a la ayuda de esta paleta.



La pestaña Page de la paleta permite editar la página que se está visualizando.

Seleccionar la page template a aplicar.

Aprobación y publicación de la página.

Creación de subcanales seleccionando una page template.

El icono  permite acceder al Workspace de los Sites. Todas las operaciones que se realizan a través del Workspace del Site también se pueden realizar des de VCM Content/vgnexttemplating.

App Instance, es la instancia del site que se está configurando.

- ***Página Secundaria:*** una página secundaria es un componente de *Portal* que determina la funcionalidad que va a tener una determinada página de *Portal.*

  Se puede modificar la apariencia de una página secundaria editando su página *JSP primaria*. P

  Vignette Portal maneja dos tipos de requests:

  - Display requests: simplemente muestran HTML
  - Process requests: ejecutan código contenido en una página secundaria y redirigen a una display request.

  Las *páginas secundarias* pueden contener clases Java, denominadas *Actions,* donde se llevan a cabo determinados procesos. Hay dos tipos de *Actions*.

  - Pre-Display Actions: se ejecutan en una *Display Request*, antes de que que la JSP de la *pagina secundaria* sea cargada
  - Process Actions: ejecutan acciones que no implican mostrar nada por pantalla. Una vez estas acciones se han llevado a cabo se redirige a una *Display Request.*

  Las *Acciones* de una *página secundaria* se describen en un fichero *component.xml* para cada *página secundaria* y son invocadas secuencialmente en el orden en que aparecen en dicho fichero.

- ***Tipo de Página Secundaria:*** al igual que en el caso de los estilos y los tipos de estilo, existen tipos de página secundaria que definan una determinada funcionalidad y de los cuales las páginas secundarias son instancias.

  Las *acciones* pueden ser definidas para un *tipo de página secundaria* o para instancias concretas de *páginas secundarias.* Lo habitual es que sean los *tipos de página secundaria* los que contienen las *acciones* y que las instancias de *página secundaria* simplemente hereden esas acciones, pero una instancia de *página secundaria* puede definir su propia lista de *pre-display actions* y de *process actions*. Para esto existen dos maneras:

  - Herencia directa: por el mero hecho de ser instancia de un *tipo de página secundaria*, la instancia cuenta con las acciones de dicho tipo.
  - Herencia java: por este procedimiento se pueden crear en instancias de *páginas secundarias* clases (correspondientes a *Actions*) que extiendan aquellas del *tipo de página secundaria.*

7.2.1 Creación de Page Templates

Una plantilla de Dynamic Portal consta de dos partes: Page Model y Page Renderer.

Template/Page Model

Cada Template está representada por un único Objeto de VCM. Las templates se almacenan en Content/vgnExtTemplating/Templates.

Nomenclatura de Page Templates:

T_<nombreSite>_<nombreTemplate>

Ejemplo:
T_alumni_home ➔ se referirá a la plantilla de la home.
T_alumni_general ➔ se referirá a la plantilla general de las páginas de alumni.

Page Render

Las Page Templates de tipo Portal Page, que son las que se definirán en Dynamic Portal, se asociarán a Navigation Items como se representa en el Figura 1 (página 30).
.

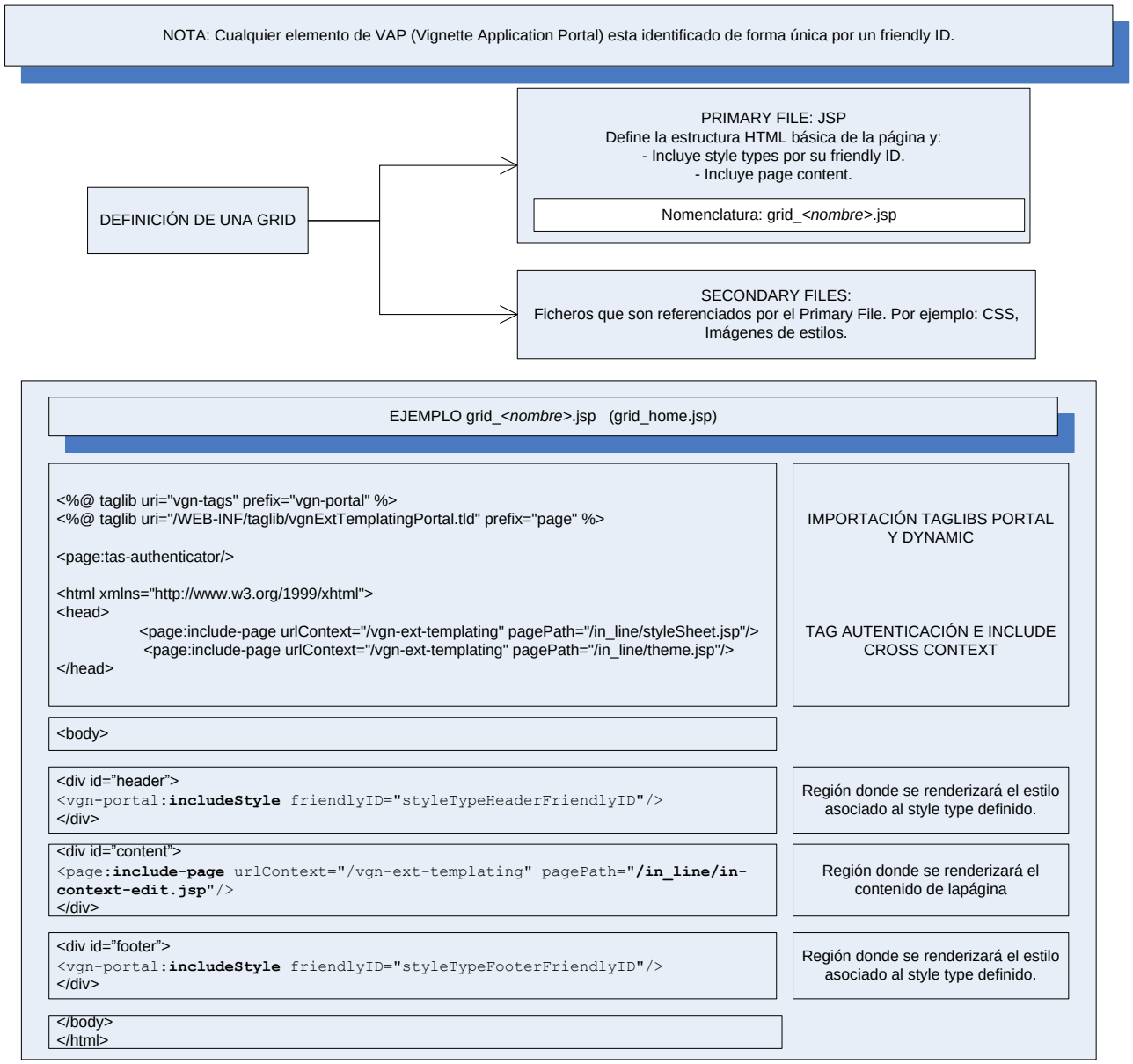
7.2.2 Personalización de la Grid

Una grid require:

- Una JSP
- Ficheros estáticos que se invocan utilizando la directiva <img> de página JSP pages in the grid.
- JSP que define los styles
- JavaScript
- Ficheros .properties para las traducciones

7.2.2.1 Ejemplo Gráfico Grid

<table>
<tr><td colspan="2">Content Region:
Cabecera, Logo corporativo
Template-scoped</td></tr>
<tr><td>Body Region Template content region, template-scoped</td><td>Body Region Page content region, page-scoped</td></tr>
</table>



7.2.2.2 Construcción de un Grid

La estructura de la Grid presentada en el esquema anterior seria:

```
<%@ page import="com.epicentric.common.Installation,
com.epicentric.common.website.ParameterConstants,
java.util.Date,
com.epicentric.common.website.EndUserUtils,
com.epicentric.template.*" %>
<%@ taglib uri="vgn-tags" prefix="vgn-portal" %>
<%@ taglib uri="/WEB-INF/taglib/vgnExtTemplatingPortal.tld"
prefix="vgndps" %>
<% long t1 = new Date().getTime(); %>
<html xmlns="<http://www.w3.org/1999/xhtml>">
<head>
```

```
<vgndps:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/styleSheet.jsp"/>
<vgndps:include-page urlContext="/vgn-ext-
templating"pagePath="/in_line/theme.jsp"/>
</head>
<body>
<vgndps:tas-authenticator/>
<vgn-portal:includeStyle friendlyID="header"/>

<vgndps:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/in-context-edit.jsp"/>

<vgn-portal:includeStyle friendlyID="footer"/>

</body>
</html>
```

- **Page Imports y Tag Libraries:**

La JSP Grid comienza con las directivas JSP para importar los paquetes Java, librerías y las taglib de Dynamic Portal

```
<%@ page import="com.epicentric.common.Installation,
com.epicentric.common.website.ParameterConstants,
com.epicentric.common.website.EndUserUtils,
java.util.Date,
com.epicentric.template.*" %>
<%@ taglib uri="vgn-tags" prefix="vgn-portal" %>
<%@ taglib uri="/WEB-INF/taglib/vgnExtTemplatingPortal.tld"
prefix="vgndps" %>
```

Se debe incluir la taglib `vgnExtTemplatingPortal.tld`.
Importante tener en cuenta que al referenciar la taglib de Dynamic se debe incluir el campo "prefix" con valor "**`vgndps`**"
Se debe incluir la taglib de portal.
Importante tener en cuenta que al referenciar la taglib de Portal se debe incluir el campo "prefix" con valor "**`vgn-portal`**"

Se deben incluir los tags de autenticación y los includes de cross-context:

```
<vgndps:tas-authenticator/>
<vgndps:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/in-context-edit.jsp"/>
<vgndps:include-page urlContext="/vgn-ext-
templating"pagePath="/in_line/styleSheet.jsp"/>
<vgndps:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/theme.jsp"/>
```

Los tags "`include-page`" para la styleSheet.jsp y el theme.jsp deben estar entre las etiquetas "`<head>`" y `</head>`.

- **Creción de regiones**

Se van a crear regiones con la sentencia “request.setAttribute”. Permite pasar parámetros al tag “contentRegion” utilizando los style types.

El siguiente código crear una region `Body Region Template` con una JSP de renderización:

```
request.setAttribute("com.vignette.ext.templating.portlet.region.name",
"Body Region Template");
%>
<%
request.setAttribute("com.vignette.ext.templating.portlet.region.jspOverr
ideUri", "/mywebapp/overrides/jspoverride.jsp");
%>

<vgn-portal:includeStyle friendlyID="mytemplatecontentregion"/>
```

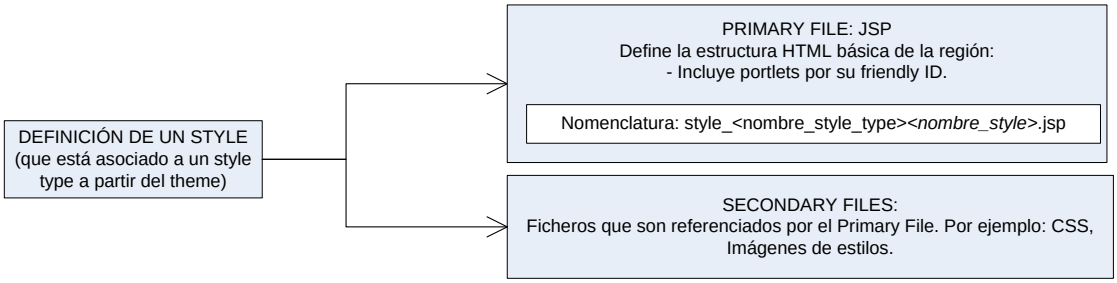
Cada región debe tener un nombre único.

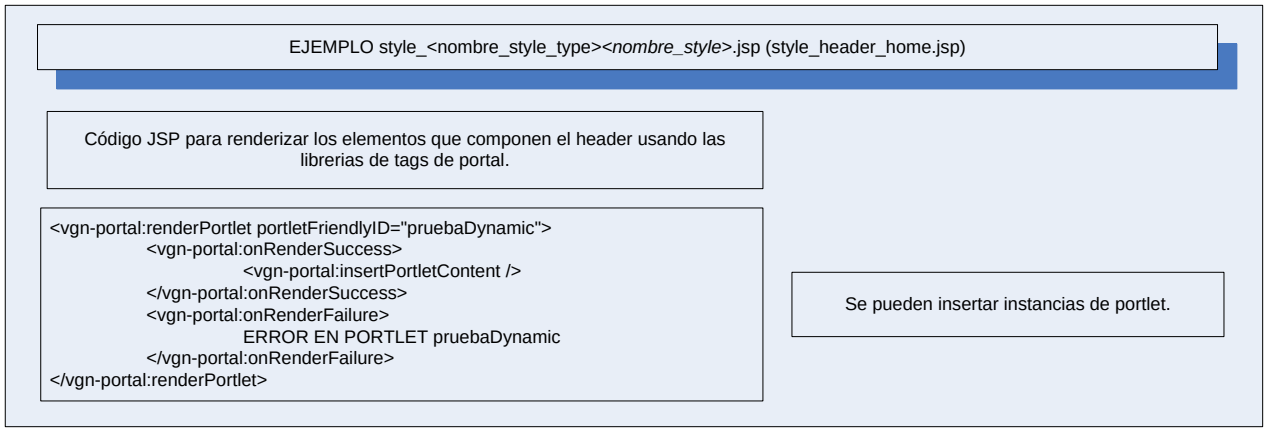
- **Style Types**

La grid ejemplo referencia styles types mediante el atributo friendly ID:

```
<vgn-portal:includeStyle friendlyID="mytemplatecontentregion"/>
<vgn-portal:includeStyle friendlyID="mypagecontentregion"/>
```

7.2.3 Desarrollo de Styles y Style Types





Se describe a continuación los dos styles definidos en el Grid del punto anterior

Se debe definer el Style Type si se quieren crear styles personalizados

- **Style Types**

Un Style Type es una colección de styles. El atributo “Friendly ID” que permite referenciar un style type desde la grid.
Cada Style type require un Friendly ID que se especifica en el fichero: `component.xml` de la siguiente manera:

```
friendly-id="string"
```

Si se crea el Style type desde la consola de VAP, se introduce el “Friendly ID” en un campo del formulario de creación, en vez de en el fichero `component.xml`

En el ejemplo losFriendly ID’s de los styles y Styles Types son:
`mytemplatecontentregion`
`mypagecontentregion`

El Style Type y Style pueden tener el mismo Friendly ID porque en el ejemplo cada Style type tiene un único Style asociado.

- **Style** `mytemplatecontentregion`

Éste Style define una template-scoped content region:

```
<%--
Copyright 1999-2009 Vignette Corporation.
All rights reserved.
THIS PROGRAM IS CONFIDENTIAL AND AN UNPUBLISHED WORK AND TRADE
SECRET OF THE COPYRIGHT HOLDER, AND DISTRIBUTED ONLY UNDER RESTRICTION.
EXCEPT AS EXPLICITLY STATED IN A WRITTEN AGREEMENT BETWEEN THE PARTIES,
THE SOFTWARE IS PROVIDED AS-IS, WITHOUT WARRANTIES OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT,
PERFORMANCE, AND QUALITY.
--%>
<%@ page import = "com.epicentric.portalbeans.*,
com.vignette.ext.templating.portlets.RegionBean,
com.vignette.portal.util.StringUtils,
java.util.StringTokenizer"
contentType = "text/html; charset=UTF-8" %>
<%@ taglib uri="/WEB-INF/taglib/vgnExtTemplatingPortal.tld"
prefix="vgndps" %>
// Enables in-context edit
request.setAttribute("com.vignette.ext.templating.portlet.region.
displayInContextEdit", "true");
// Sets the region scope to template.
String scope = RegionBean.SCOPE_TEMPLATE;
request.setAttribute("com.vignette.ext.templating.portlet.region.scope",
scope);
%>
<vgndps:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/contentRegion.jsp"/>
```

El Style ejemplo empieza con las siguientes sentencias:

```
<%@ page import = "com.epicentric.portalbeans.*,
com.vignette.ext.templating.portlets.RegionBean,
```

```
com.vignette.portal.util.StringUtils,
java.util.StringTokenizer"
contentType = "text/html; charset=UTF-8" %>
<%@ taglib uri="/WEB-INF/taglib/vgnExtTemplatingPortal.tld"
prefix="vgndps" %>
```

Se debe referenciar a la TagLib de DynamicPortal `vgnExtTemplatingPortal.tld`. Al referenciarla la se debe añadir el atributo "prefix" con valor `vgndps`.

De la misma manera que en la personalización de Grids, los Styles utilizan la sentencia `request.setAttribute` para habilitar el modo " incontex editing" y establecer el scope de la región.

Cada Style que defina una content region debe llamar a la JSP `contentRegion.jsp`:

```
<prefix:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/contentRegion.jsp"/>
```

- **Style mypagecontentregion**

La region permite crear una page-scoped content region de la siguiente manera:

```
<%--
Copyright 1999-2009 Vignette Corporation.
All rights reserved.
THIS PROGRAM IS CONFIDENTIAL AND AN UNPUBLISHED WORK AND TRADE
SECRET OF THE COPYRIGHT HOLDER, AND DISTRIBUTED ONLY UNDER RESTRICTION.
EXCEPT AS EXPLICITLY STATED IN A WRITTEN AGREEMENT BETWEEN THE PARTIES,
THE SOFTWARE IS PROVIDED AS-IS, WITHOUT WARRANTIES OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT,
PERFORMANCE, AND QUALITY.
--%>
<%@ page import = "com.epicentric.portalbeans.*,
com.vignette.ext.templating.portlets.RegionBean,
com.vignette.portal.util.StringUtils,
java.util.StringTokenizer"
contentType = "text/html; charset=UTF-8" %>
<%@ taglib uri="/WEB-INF/taglib/vgnExtTemplatingPortal.tld"
prefix="vgndps" %>
// Enables in-context edit
request.setAttribute("com.vignette.ext.templating.portlet.region.
displayInContextEdit", "true");
// Sets the region scope to template.
String scope = RegionBean.SCOPE_PAGE;
request.setAttribute("com.vignette.ext.templating.portlet.region.scope",
scope);
%>
<vgndps:include-page urlContext="/vgn-ext-templating"
pagePath="/in_line/contentRegion.jsp"/>
```

Éste Style es prácticamente igual al Styl e mytemplatecontentregion excepto por: `RegionBean.SCOPE_PAGE` en vez de `RegionBean.SCOPE_TEMPLATE`.

7.2.4 Desarrollo de Display Views

- Dynamic Portal proporciona una serie de componentes por defecto (Navigation, Content Item, Smart List, etc.) que simplifican la construcción de una web. Los más destacados son:
  - Navigation Component: Muestra el árbol de navegación, migas de pan…
  - Smart List Component: Permite configurar una consulta de elementos que se recuperarán del VCM.
- Cada región de contenido de una página, que se representa mediante un Portlet Content Publisher 2.0 puede tener uno o más componentes.
- La JSP es almacenada en una instancia del objeto Display View.
- La salida de una Display View es cacheada, se debe configurar el TTL (Time-To-Live) para determinar la frecuencia de cacheado de la región.
- Las Display Views son utilizadas para renderizar los styles de un componente.
- Cada tipo de componente puede tener una o más displays views.
- El desarrollador asocia un render con cada display view con el objetivo de crear la salida apropiada.
- Un usuario de negocio puede elegir la vista deseada para cada componente.
- Las Display View simplifican la presentación sin necesidad de saber que una JSP es la que realiza la presentación.
- Se explica a continuación cómo crear JSP de renderización con Dynamic Portal. Se puede utilizar una JSP para personalizar la apariencia de un content item o un content component ítem, en un región.
- Las JSP Renders pueden ser específicas por region, es possible mostrar el mismo contenido de diferentes maneras utilizando distintos renders en diferentes regions.
- (Por defecto si no se especifica un render, un content item o content component se muestra como viene definido en la XSL de preview)
- La salida de JSP renders está cacheada

7.2.4.1 Arquitectura para Display Views

La arquitectura ie-framework proporciona una implementación básica de Managers que permite encapsular operaciones habituales sobre cada tipo de componente usado en una Display View.

A continuación se presentan los tipos de componentes más usados y se explican los Managers proporcionados por la arquitectura:

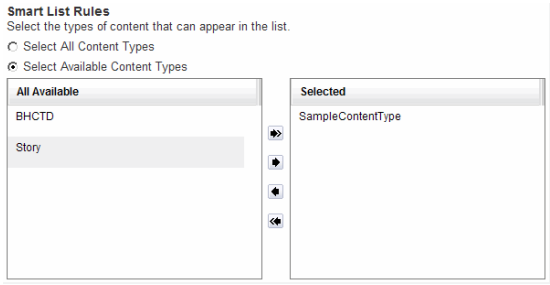
Componente	Manager de ie-framework
Smart List	Ie.managers.QueryComponentManager
Channel Navigation	*
Content Select	Ie.managers.ContentSelectComponentManager

7.2.4.1.1 Smart List y ie.managers.QueryComponentManager:

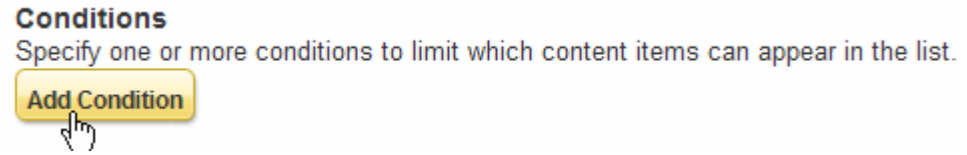
Permite recuperar una lista de instancias de contenido dinámica definiendo filtros de búsqueda.

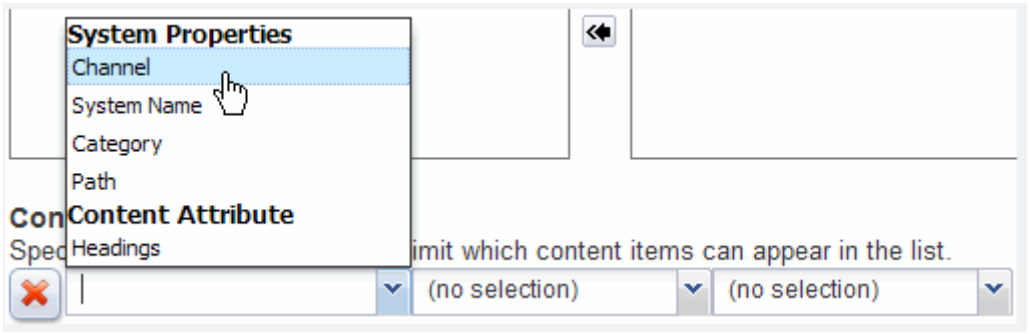
Filtros de búsqueda:

- Definir el tipo de contenido a recuperar



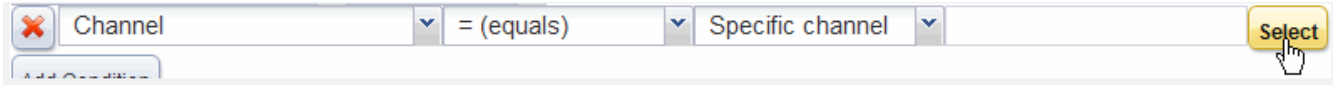
- Definir criterios de la búsqueda





Permite definir condiciones de búsqueda por:

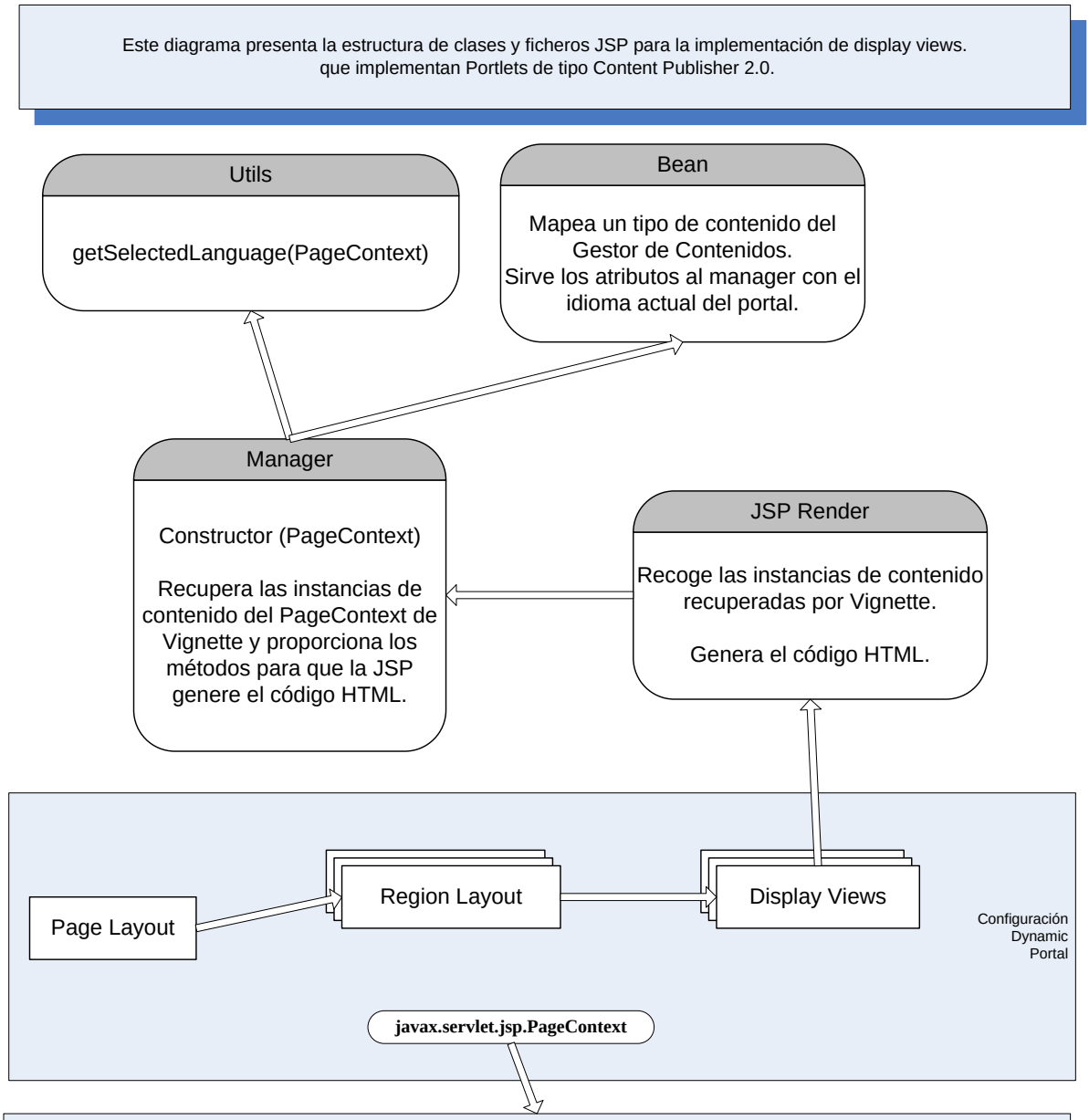
- Nombre
- Canal asociado
- Categoria asociada



QueryComponentManager devuelve la lista de instancias recuperadas por la búsqueda en la variable “contenidosDePagina”. Esta variable contiene las clases instanciadas y con todos los atributos informados y el selectedLanguage informado con el idioma actual del portal.

7.2.4.1.2 Channel Navigation:

Permite definir una región de navegación por canales.



The Vignette Portal framework includes two interfaces that represent requests and responses between the client and the portal: `com.vignette.portal.website.enduser.PortalRequest` and `com.vignette.portal.website.enduser.PortalResponse`, respectively. You can access `PortalRequest` and `PortalResponse` through another interface in the same package: `com.vignette.portal.website.enduser.PortalContext`. Every Vignette Portal JSP has access to `PortalContext` as an implicit JSP object named `portalContext`.

You retrieve the `PortalRequest` from the `PortalContext` as follows:

```
PortalRequest portalRequest = portalContext.getPortalRequest();
```

You can then retrieve a specific request parameter as follows:

```
String myParam = portalRequest.getParameter(MY_PARAM_NAME);
```

(Vignette_Portal_8.0_Developer_s_Guide.pdf)

7.2.4.2 Ejemplo Plantilla Display View

- Una JSP Display View empieza declarando la TagLibrary de Dynamic Portaltag y la directiva `page`:

```
<%-- declarations --%>
<%@taglib uri="/WEB-INF/vgnExtTemplating.tld"
prefix="templating" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
<%@ page contentType="text/html" %>
```

- Se inicializa el componente y se declara una variable para almacenar el bean del componente:

```
<templating:initComponent var="component"/>
```

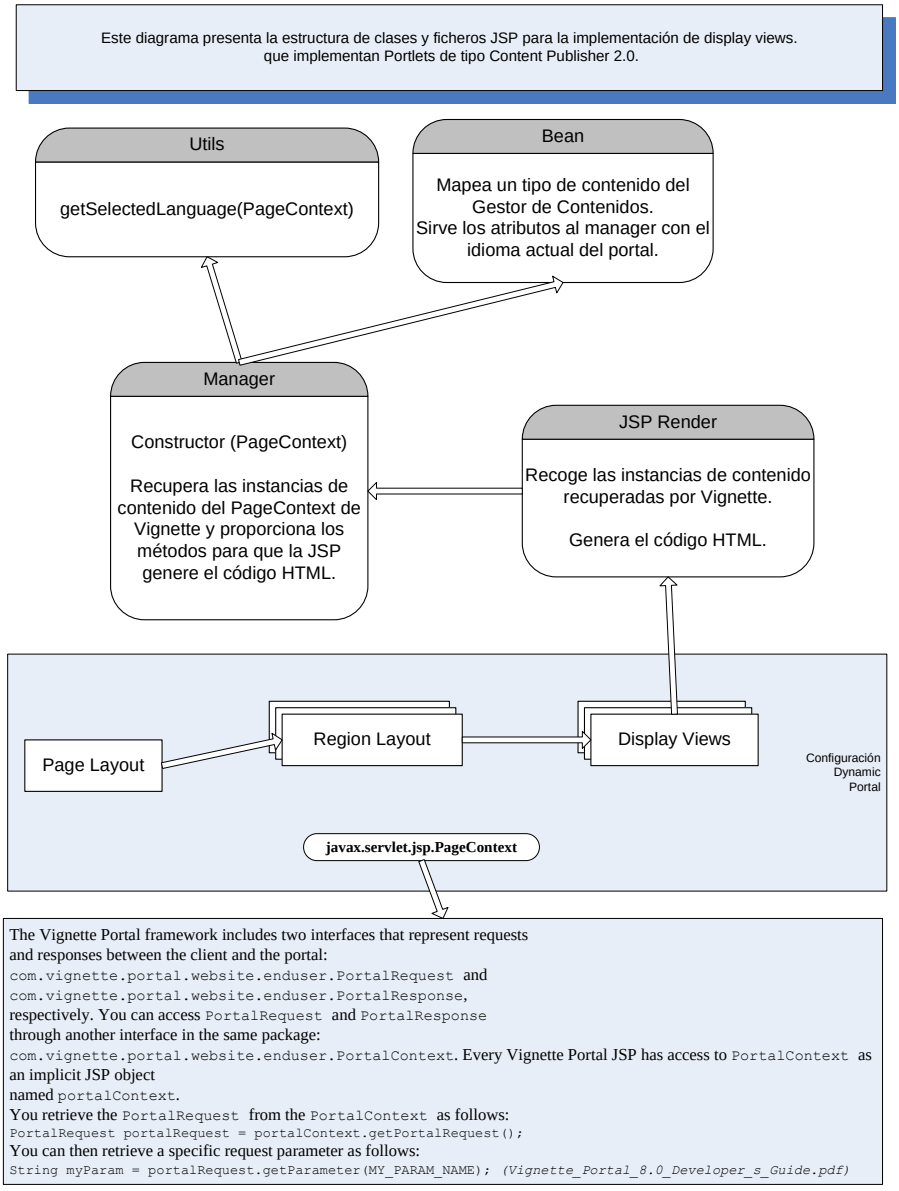
- Se accede a los atributos del component mediante JSTL.
- El siguiente ejemplo mostrará el título del component, el header, body y footer:

```
<c:if test="${not empty component.title}">
<h1 class="contentTitle">${component.title}</h1>
</c:if>
<c:if test="${not empty component.header}">
<c:out value="${component.header}" escapeXml="false"/>
</c:if>
<c:if test="${not empty component.text}">
<div><c:out value="${component.text}" escapeXml="false"/>
</div>
</c:if>
<c:if test="${not empty component.footer}">
<c:out value="${component.footer}" escapeXml="false"/>
</c:if>
```

- El siguiente fragmento es un ejemplo de cómo mostrar una lista de enlaces a contenido en canal:

```
<templating:initRequestContext var="rc"/>
<c:set var="currentChannel"
value="${rc.requestedChannelBean}"/>
<c:if test="${not empty component.title}">
<h1 class="componentTitle">${component.title}</h1>
</c:if>
<c:out value="${component.header}" escapeXml="false"/>
<ul class="linklist">
<c:forEach items="${component.results}" var="content">
<templating:contentLink var="linkUrl"
channelId="${currentChannel.systemId}"
oid="${content.systemId}"/>
<li class="content">
<a href="${linkUrl}">
${content.title}
</a>
</li>
</c:forEach>
</ul>
```

Para una mejor gestión de los componentes de la plantilla, se utilizará una clase Java llamada Manager que preparará el contenido recuperado por el componente y que debemos mostrar en la página. En función del tipo de componente definido se usará un manager específico como muestra el siguiente esquema:



A más bajo nivel, tendremos la relación con los managers que proporciona la arquitectura para cada región.

7.2.4.3 Registro Display View

Las Display Views deben ser registradas en el Workspace de presentación

Al registrar las Display Views se debe especficar una “app instance”. Las Display Views aparecen como una opción de la lista de selección en el entorno de preview, solo cuando se ejecuta la “app instance” especificada

Las Display Views deben estar compartidas con los sites y asociadas a los component types

Estas tareas se describen el la ayuda de Content Workspaces y en la guía *Vignette Dynamic Portal Module and Vignette Dynamic Site Module Administration Guide.*

7.2.5 Resumen

1. Acceder a la consola de VAP y crear la grid, style y style types
2. Importar los componentes personalizados utilizando un fichero .care.
3. Desplegar en el site de VAP los components antes importados (más información en *VAP Developer's Guide)*
4. Compartir la grid, styles types y styles con el site de VAP que se utilizar con Dynamic Portal
5. *Opcional*:
    - Acceder a la consola de Administración de VAP del site que se quiere desarrollar con Dynamic Portal
    - Cambiar la grid por defecto por la grid desarrollada
6. Crear las páginas de VAP y enlazarlas con los navigation items

Crear una page template y asociarla a los navigation items creados en el punto anterior

8 Desarrollos en Vignette: Creación de portlets

Se define *portlet* como una aplicación web gestionado por un contenedor que general que procesa requests y presentan contenidos dinámicamente en el interfaz del usuario del portal. En el contexto de Vignette, los Portlets se situan en las diversas páginas de Portal definidas en sitios específicos para realizar todo tipo de funciones, como puede ser mostrar contenidos.

A la hora de desarrollar Portlets se pueden desarrollar de acuerdo con el estándar JSR 168 o implementar *PortalBeans,* un mecanismo propietario de Vignette para el desarrollo y despliegue de Portlets.

8.1 Desarrollo de Portlets JSR 168

Una de las características principales de los Portlets es que tienen una serie de modos de funcionamiento (*Portlet Modes*) y de estados de la ventana (*Window State*).

Modo de funcionamiento: El modo de funcionamiento indica la función que el Portlet está llevando a cabo en un momento determinado. Para los Portlets JSR 168 los modos definidos son *View*, *Edit* y *Help*. Estos modos son usados por el método *render()* por defecto del Portlet para determinar qué método de visualización invocar. Más adelante se especificarán los métodos básicos implementados por los Portlets por defecto.

Adicionalmente se pueden definir modos custom para desempeñar funciones específicas en la medida en que sea necesario. Estos modos se deben declarar en el fichero *web.xml.*

Estado de la ventana: El estado de la ventana nos indica la cantidad de espacio en una página de *Portal* que será asignada al Portlet. El Portlet puede usar esta información para determinar qué información mostrar. Los estados de ventana básicos son *minimizado*, *maximizado* y *normal.*

Al igual que con los modos del Portlet, se pueden definir estados de la ventana a medida. Estos estados de ventana custom deben ser declarados en el fichero *web.xml.*

Container: la especificación de Portlets JSR 168 define un Container que manejará los Portlets. Un Container consiste en un componente que proporciona a los Portlets un entorno de ejecución, gestiona su ciclo de vida y les proporciona almacenamiento persistente. El Container no es responsable , sin embargo del contenido a mostrar por el Portlet.

Para este Container se define un contrato por el cual el Portlet debe implementar los siguientes métodos:

- init(): se invoca cuando el Container instancia el Portlet. Contiene la lógica necesaria para preparar al Portlet para atender peticiones.
- destroy(): se invoca cuando el container destruye el portlet. Contiene la lógica necesaria para efectuar la limpieza cuando el Portlet ya no es necesitado.
- processAction(): se invoca cuando el usuario envía alguna petición de algún tipo al Portlet.
- render(): se invoca cuando el Portlet es mostrado.

Además de estos métodos, que son invocados directamente por el Container, existe una clase llamada *GenericPortlet* que implementa el método *render()* y en él se delega esta llamada en métodos más específicos dependiendo de cuál sea el estado del Portlet en el momento de esa llamada. El desarrollador puede extender esta clase e implementar, de estos métodos más específicos los que considere oportunos.

- doView(): invocado dentro del método *render()* cuando el Portlet se encuentra en modo *View, d*efine las acciones a realizar al cargar la página vista *View* del Portlet.
- doEdit():invocado dentro del método *render()* cuando el Portlet se encuentra en modo *Edit, d*efine las acciones a realizar al cargar la página vista de edición del Portlet.
- doHelp():invocado dentro del método *render()* cuando el Portlet se encuentra en modo *Help, d*efine las acciones a realizar al cargar la página vista de ayuda del Portlet.

PortletRequest y PortletResponse: Los métodos *processAction(), render()* y aquellos métodos más específicos invocados por *render()* manejan objetos *PortletRequest* y *PortletResponse* a fin de recibir inputs del usuario (a través de formularios) , comunicar con otros Portlets, JSPs o Servlets, crear el contenido que va a ser mostrado y que se envía al usuario a través del objeto *response*.

Asimismo, los Portlets habitualmente implementan vistas customizadas o diferentes comportamientos según el usuario. Esta clase de personalizaciones se implementan a través de una serie de pares *propiedad-valor* llamadas *preferencia*. La especificación JSR 168 define el interfaz *PortletPreferences,* que posee una serie de métodos tipos *getValue()* y *setValue()* mediante los cuales el Container puede recuperar y guardar esas preferencias. Un Portlet únicamente puede modificar una preferencia durante una invocación a *processAction()* . Antes de la finalización de este método se debe invocar al método *store()* para hacer los cambios permanentes.

Los Portlets JSR 168 se empaquetan y despliegan como una aplicación web estándar (ficheros WAR). Además del descriptor *web.xml* presente en cualquier fichero WAR, existe adicionalmente el fichero *portlet.xml* que define a los portlets y todos los datos de configuración relacionados.

8.1.1 Estructura de directorios y ficheros de un Portlet

Vistas: a la hora de mostrar información, el Portlet emplea una serie de JSPs que mostrarán la información pertinente en cada momento según el modo en que se encuentre el Portlet. Existe una librería de Tags específica que proporciona acceso desde la JSP a objetos específicos de los Portlets como pueden ser RenderRequest, RenderResponse, PortletConfig, PortletPreferences, etc.

Para usar la librería de TAGS es necesario hacer la siguiente declaración:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
```

En esta librería contamos con los siguientes *tags*:

- **defineObjects:** define los objetos RenderRequest, RenderResponse y PortletConfig.
- **actionURL:** este Tag crea un objeto URL que apunta al presente Portlet y desencadena una acción con los parámetros especificados. Añadiéndole una serie de atributos a la Tag se pueden añadir parámetros a la URL. Los atributos que se pueden añadir son los siguientes:
    - **windowState:** indica el estado en el que debe estar la ventana del portlet cuando el enlace sea ejecutado. Si el estado actual no es el especificado se lanza una JSPException.
    - **portletMode:** indica el modo en el que debe estar el portlet cuando el enlace sea ejecutado. Si el estado actual no es el especificado se lanza una JSPException.
    - **var:** permite especificar cualquier variable para ser transimitida vía parámetro por la URL.
    - **secure:** indica si la URL resultante debería ser para una conexión segura (secure="true") o si no (secure="false"). Si el valor especificado no está implementado por el portal se lanza una JspException.
- **renderURL:** similar a *actionURL* pero en este caso
- **namespace :** este tag proporciona un valor único para el Portlet, evitando así conflictos por coincidencia con los nombres de otros elementos del portal.
- **param:** define un parámetro o atributo que puede ser añadido a una renderURL o actionURL. Se debe especificar el nombre del parámetro y su correspondiente valor.

Las JSPs del Portlet deben ser incluidas en el directorio **/WEB-INF/jsp/** dentro del .WAR que empaqueta la aplicación.

Clase principal: un Portlet JSR 168 puede contener una clase principal que debe heredar de la clase *GenericPortlet.* Esta clase debe implementar una serie de métodos.

- *init(): e*ste método es invocado por el container antes de que el portal comience a mandarle requests.
- *doView():* suponiendo que la inicialización se halla efectuado sin incidencias y que el Portlet esté en modo VIEW, el container invoca a este método para empezar a mostrar contenido. Lo habitual es que, además de la lógica que se tenga que llevar a cabo para ejecutar el método, se especifique para el objeto *renderResponse* el tipo de contenido que se va a devolver y el finalizar se redirija a la vista que vaya a ser mostrada.
- *doEdit():* seguirá exactamente la misma lógica que en el caso del método *doView* salvo que este método será invocado cuando el Portlet esté en modo EDIT.
- *doHelp():* cuando el usuario pulsa el botón *help* en la barra de título del Portlet, este cambia a modo HELP y el método *doHelp()* es invocado. Este método sigue la misma lógica que *doView()* y *doEdit().*

A continuación se muestra un ejemplo de clase principal con lo mínimo para poder ser ejecutada.

```
public class UnPortlet extends GenericPortlet
{
	public void init(PortletConfig config) throws PortletException
	{
		super.init(config);
		//Aquí iría la lógica necesaria para inicializar el Portlet
	}
	public void doView(RenderRequest rReq, RenderResponse rRes) throws PortletException
	{
		rRes.setContentType("text/html");
		try
		{
			//Aquí iría la lógica necesaria para mostrar la vista principal del Portlet
			PortletRequestDispatcher rd =
			getPortletContext().getRequestDispatcher("/../unPortletview.jsp");
			rd.include(rReq,rRes);
		}
		catch(Exception ex)
		{
			//Si se produjese una excepción se redirigiría a una página de error
			PortletRequestDispatcher rd =
			getPortletContext().getRequestDispatcher("/../serviceUnavailable.html");
			rd.include(rReq,rRes);
		}
	}
	public void doEdit(RenderRequest rReq, RenderResponse rRes) throws PortletException
	{
		rRes.setContentType("text/html");
		String error = "";
		//Aquí iría la lógica necesaria para mostrar la vista de edición del Portlet
		if(error != null)
		{
			error = "<font color=\"red\"><b>ERROR: </b>" + error + "</error>";
		}
		else
		{
			error = "";
		}
		rReq.setAttribute("error",error);
		PortletRequestDispatcher rd =
		getPortletContext().getRequestDispatcher("/../unPortletEdit.jsp");
		rd.include(rReq,rRes);
	}
	public void doHelp(RenderRequest rReq, RenderResponse rRes) throws PortletException
	{
		rRes.setContentType("text/html");
		PortletRequestDispatcher rd =
		getPortletContext().getRequestDispatcher("/../unPortletHelp.jsp");
		rd.include(rReq,rRes);
	}
```

Como se puede apreciar, al final de cada método se crea un objeto PortletRequestDispatcher que redirige al usuario a la vista correspondiente. En este objeto se han de incluir los objetos correspondientes a la Request y Response a fin de que los parámetros incluidos en estas puedan estar disponibles en las JSPs correspondientes a las vistas.

Si el usuario envía información vía formulario se ejecuta el método processAction():

```
public void processAction(ActionRequest aReq,ActionResponse aRes) throws portletException
{
        boolean editOK;
        String errorMsg;
        PortletPreferences preferencias = aReq.getPreferences();

        if(aReq.getPortletMode().equals(PortletMode.VIEW))
        {
                //aquí se ejecutan las acciones correspondientes cuando el Portlet esté en modo
                //VIEW
                aRes.setRenderParameter("nombreParametro",valor);
        }
        if(aReq.getPortletMode().equals(PortletMode.EDIT))
        {
                //aquí se ejecutan las acciones correspondientes cuando el Portlet esté en modo
                //EDIT
                preferencias.setValue("nombreParametro1",valor);
                preferencias.setValue("nombreParametro2",valor);
                try{
                        preferencias.store();
                        editOK = true;
                }
                catch(Exception ex){
                        editOk = false;
                        error = ex.getMessage();
                }
                //Los valores que el usuario introduzca se guardan en el objeto
                //PortletPreferences.

                if(editOk){
                        aRes.setPortletMode(PortletMode.VIEW);
                }
                else{
                        aRes.setRenderParameter("error",error)
                }
        }
        …
}
```

Las clases que vaya a llevar el Portlet se deben incluir en el directorio **/WEB-INF/clases/*classpath*/** donde *classpath* es el paquete de la clase en cuestión.

Portlet.xml: se trata de un fichero xml donde se detallan todas las opciones de configuración específicas para Portlets. Algunas de las opciones que se pueden especificar en este fichero son las siguientes:

- Nombre del Portlet:
  ```
  <portlet-name>UnPortlet</portlet-name>
  ```

- Parámetros de inicialización:
  ```
  <init-param>
      <name></name>
      <value></value>
  </init-param>
  ```

- Clase principal del Portlet (si la hubiere):

```
<portlet-class>com.sun.portal.portlet.samples.UnPortlet</portlet-class>
```

- Modos de funcionamiento del Portlet:

```
<supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>EDIT</portlet-mode>
    <portlet-mode>HELP</portlet-mode>
</supports>
```

- Preferencias del Portlet:

```
<portlet-preferences>
        <preference>
                <name>contentPage</name>
                <value>/unPortlet/content.jsp</value>
        </preference>
        <preference>
                <name>editPage</name>
                <value>/unPortlet/edit.jsp</value>
        </preference>
</portlet-preferences>
```

Debe estar situado en /WEB-INF/porlet.xml

Web.xml: fichero xml común a cualquier aplicación web que describe cómo debe ser ésta desplegada.

Debe estar situado en /WEB-INF/porlet.xml.Ejemplo:

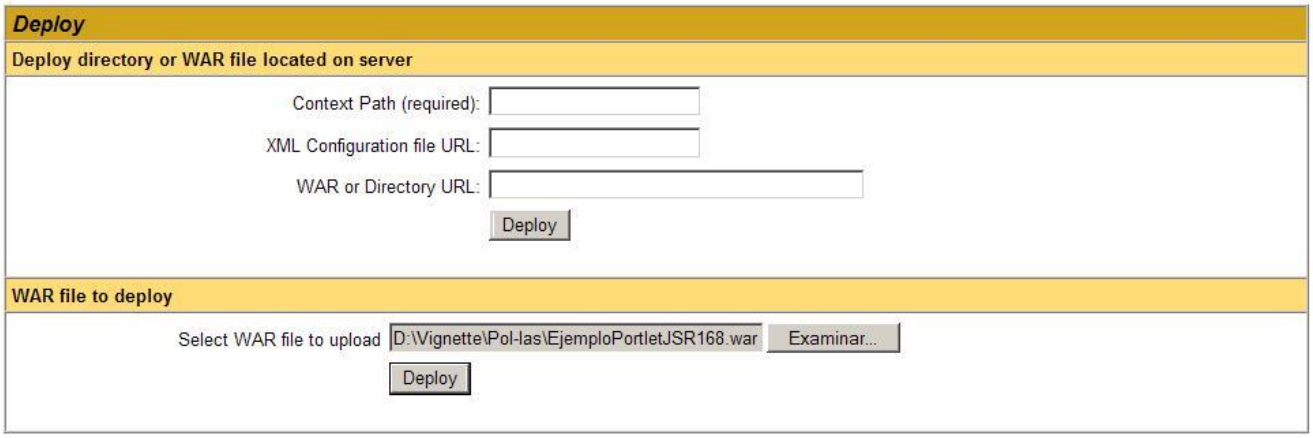
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"file:/etc/opt/SUNWps/dtd/web-app_2_3.dtd">

<web-app>
    <display-name>Sample Portlet Application</display-name>
</web-app>
```

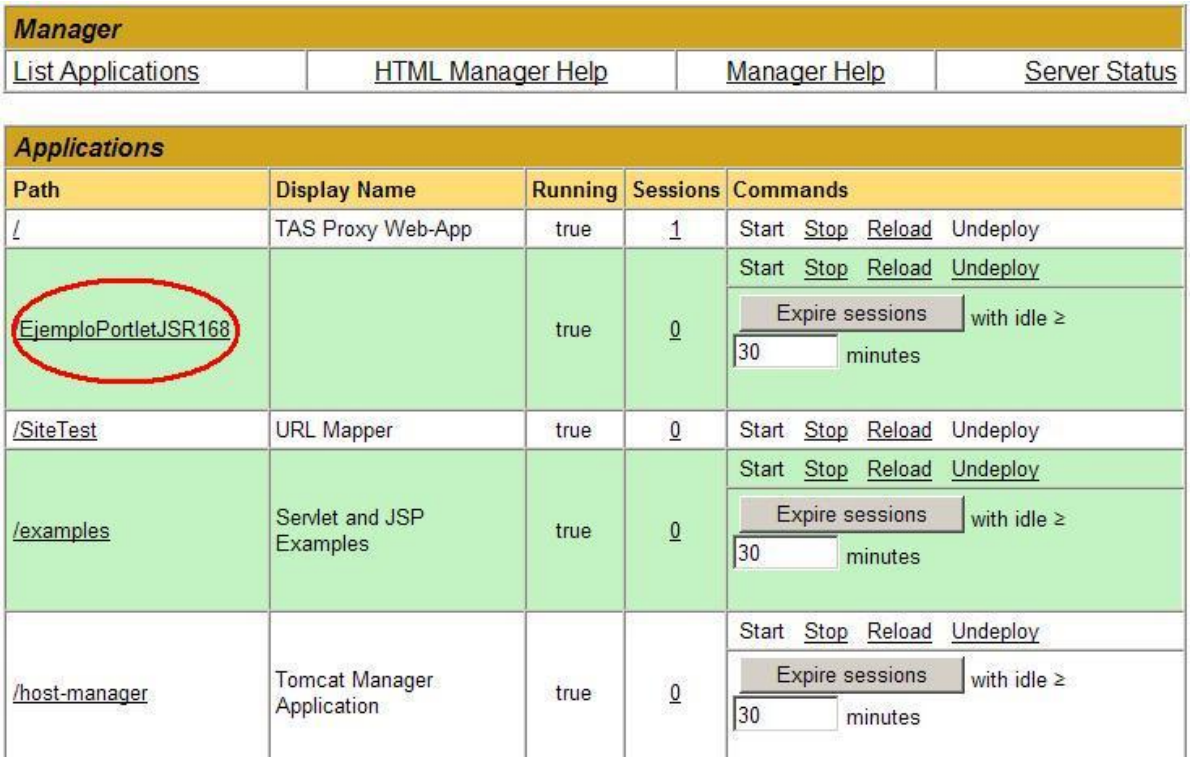
8.1.2 Despliegue de un Portlet JSR 168 en Vignette Portal

Para desplegar un Portlet a fin de usarlo en Portal en primer lugar es necesario comprimir en un fichero .WAR los ficheros del Portlet siguiendo la estructura de directorios anteriormente descrita.

Un vez creado el .war accedemos a la consola de Tomcat y en la sección de despliegue seleccionamos el fichero .war y damos a desplegar:



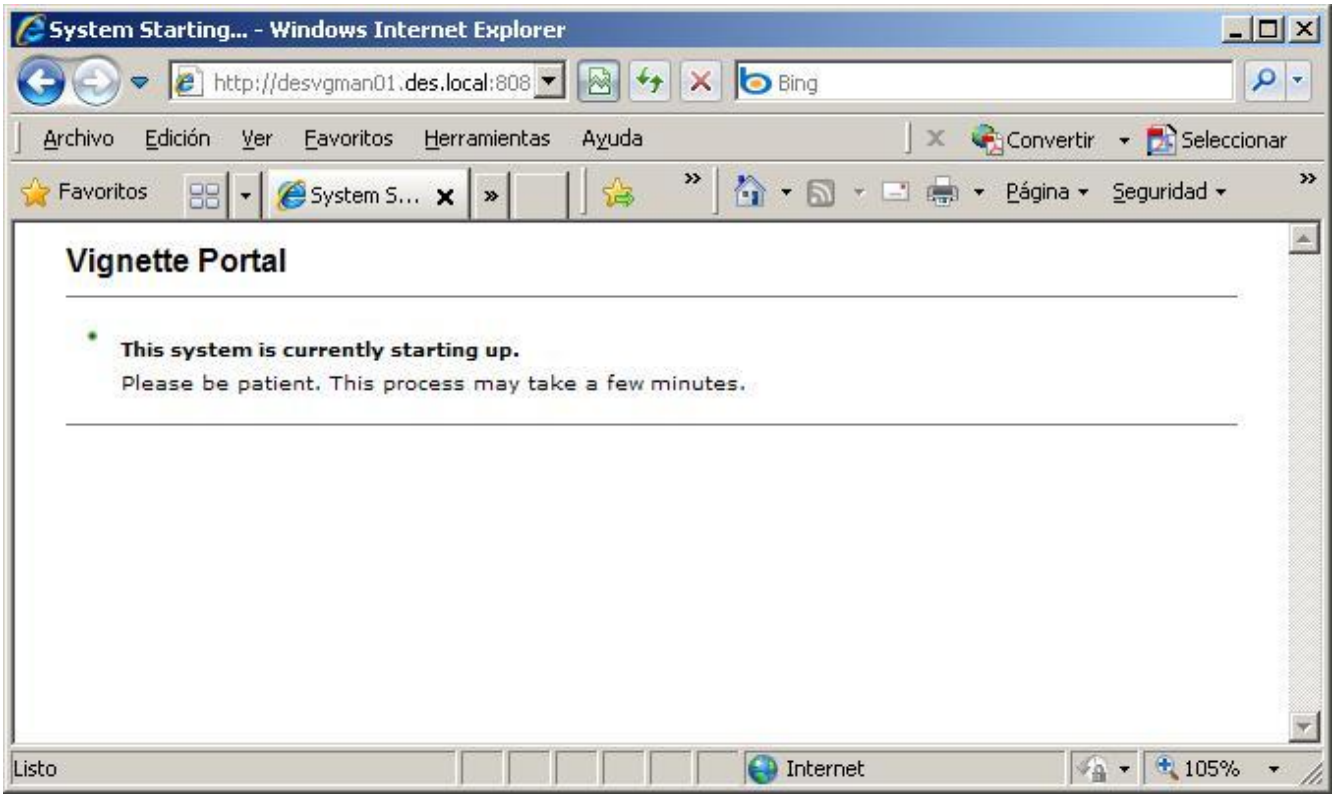
Si todo funciona correctamente, Tomcat nos mostrará, en el listado de aplicaciones ejecutándose en el servidor el portlet corriendo:



Una vez desplegado es necesario detener Vignette Portal y volver a arrancarlo. Para ello accedemos a la administración de Tomcat y en ella detenemos la aplicación web *portal* y la volvemos a arrancar.

/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/portal	Vignette Portal	true	2	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/sample	Hello, World Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/vgn-community-applications	Community Application Portlets	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Portal tarda unos instantes en arrancar. Si en el momento en que está arrancando intentamos acceder, veremos la siguiente patalla.



8.2 Desarrollo de PortalBeans

Un PortalBean es una aplicación web que puede ser desplegada en Vignette Portal y empleada en sites. El entorno Vignette Portal propociona el *container* en el que funciona el PortalBean y proporciona un look & feel unificado, de forma que el PortalBean es únicamente reponsable del contenido a mostrar. Vignette Portal presenta los PortalBeans al usuario final dentro de páginas de Portal, su apariencia estará controlada mediante *estilos* y su posicionamiento en la página es controlado por la *grid.*

Los Portalbeans reciben dicho nombre porque implementan la interfaz *com.epicentric.portalbean.PortalBean* (ya que han de extender la clase *GenericPortalBean* o una de sus subclases). Cada instancia de portlet PortalBean incluye una instancia del objeto correspondiente que implementa la interfaz *PortalBean.*

Al igual que con los *estilos* existen Tipos de PortalBean, de los cuales se crean instancias concretas, que el administrador podrá situar en páginas de Vignette Portal.

Los objetos PortalBean existen durante toda la vida del portlet. Son creados cuando el portlet es instanciado y son serializados cuando Vignette Portal es detenido.

Los PortalBeans son alicaciones que siguen el patrón MVC (modelo-vista-controlador):

- Modelo: representa la lógica de negocio y la lógica del portlet y reside en la clase PortalBean y en aquellos objetos específicos del Portlet accedidos por la clase PortalBean.
- Vista: es representada por aquellas páginas JSP que representan la parte gráfica de los portlets
- Controlador: recoge los inputs del usuario a través de las vistas, modifica el estado del modelo y efectua transiciones entre vistas.

8.2.1 Estructura de directorios y componentes de un PortalBean

PortalBeanDescriptor: todo PortalBean consta de un fichero xml de extensión .pbd. Por cada .pbd que Vignette Portal halla a la hora de efectuar el despliegue en el directorio *DirectorioInstacionPortal*/WEB-INF/config/beans/defs se crea un objeto *PortalBeanDescriptor.*

Cada tipo de PortalBean tiene su propio .pbd y todas las instancias de dicho tipo comparten fichero .pbd así como el objeto *PortalBeanDescriptor* asociado a este.

La ruta donde el PortalBeanDescriptor debe hallarse es WEB-INF\config\beans\defs

A continuación se muestra un .pbd en su mínima expresión:

```
<PORTALBEAN ID="PortalBean_type"
CLASS="fully_qualified_class_name"
TYPE_UID="unique_identifier"
VERSION="PortalBean_version_number" >

        <TITLE>PortalBean_title</TITLE>
        <NAME>string_for_serializing</NAME>
        <DATA>
                <FIELD NAME="JSPDirName" VALUE="location_of_JSPs"</FIELD>
        </DATA>
        <VIEW ID="standard_view_name">
        </VIEW>

        <VIEW>
                Las especificaciones de las vistas del Portlet van aquí
        </VIEW>

</PORTALBEAN>
```

Clase PortalBean: esta clase debe implementar la interfaz *com.epicentric.portalbean.PortalBean.* Para ello se implementa la clase *GenericPortalBean* o una de sus subclases. Si el Portlet va a mostrar las vistas mediante

páginas JSP lo más recomendable es extender la clase *com.epicentric.portalbeans.beans.jspbean.JSPBean* (una subclase de *GenericPortalBean*) y sobrecargar el método *getView()* a fin de efectuar el mapeo necesario entre vistas y páginas JSP.

Para ello JSPBean proporciona una serie de métodos que se heredarán, como es *getJspFileName()* que se encarga del mapeo entre IDs de vistas y páginas JSP. Ejemplo:

```
import com.epicentric.portalbeans.PortalBean;
import com.epicentric.portalbeans.PortalBeanView;
import com.epicentric.portalbeans.PortalPageContext;
import com.epicentric.portalbeans.beans.jspbean.JSPBean;
import com.epicentric.portalbeans.beans.jspbean.JSPView;


//Se debe extender la clase JSPBean para beneficiarnos de los métodos de la superclase concernientes a
//mapeo de páginas JSP contra vistas definidas.
public class UnPortalBean extends JSPBean
{
	private final static long serialVersionID = 1L;

	public PortalBeanView getView(PortalPageContext ppc)
	{
		//A través del objeto PortalPageContext obtenemos el ID de la vista actual

		String viewID = ppc.getViewID();


		//A través del método getJspFilename de la superclase (JSPBean), proporcionando el ID de
		//la vista se obtiene el nombre de la JSP que mapea contra esa ID
		String jspFile = getJspFilename(viewID,ppc.getDocumentType());


		//Según de qué vista de las definidas en el .pbd se trate, redirigimos a una JSP u otra.
		//Para ello, se crea un objeto de tipo UserInfoCommonView o JSPView, según se trate o no
		//de una vista estándar
		if (PortalBeanView.MY_PORTAL_VIEW.equals(viewID))
		{
			return new UserInfoCommonView(this, ppc, jspFile);
		}
		else if (PortalBeanView.USER_BEAN_EDIT_VIEW.equals(viewID))
		{
			return new UserInfoCommonView(this, ppc, jspFile);
		}
		else
		{
			return new JSPView(this, ppc, jspFile);
		}
	}
}
```

Vistas: una vista es un conjunto de información que el usuario va a ver en el PortalBean presentada de una manera determinada y con una serie de posibilidades de interacción.

Las vistas que el PortalBean va a tener se definen en el fichero .pbd.

Cada vista habitualmente llevará asociada un fichero .jsp, donde se define cómo se mostrará la información al usuario en términos de presentación.

Adicionalmente cada vista puede llevar asociada una clase que implementará la interfaz *PortalBeanView* extendiendo la clase *com.epicentric.portalbeans.beans.jspbean.JSPView.* En esta clase se implementarán los métodos que sean necesarios para presentar información al usuario y que estarán disponibles desde la JSP.

De acuerdo con su funcionalidad las vistas se pueden clasificar en :

- Display views: generan contenido que el usuario verá en la página final.
- Process views: no aparecen en páginas finales, sino que procesan información.

De acuerdo con su naturaleza existe la siguiente clasificación:

- Estándar: vistas predefinidas (tienen un ID predefinido) en el entorno de Vignette Portal. Ejemplos serían la vista principal o la vista de edición de un PortalBean.
- No estándar: son vistas definidas a medida para cada PortalBean y que tienen una funcionalidad concreta.

Como se veía anteriormente las vistas se deben definir en el fichero .pbd. Las vistas estándar se definirían en el .pbd de la siguiente manera:

```
<VIEW ID="MY_PORTAL_VIEW" >
```

De esa forma ya le indicamos a Portal que el PortalBean va a usar la vista estándar **VIEW**.

En la siguiente tabla se muestran las diversas vistas estándar implementadas por Portal:

Nombre	Propósito	JSP predefinida
MY_PORTAL_VIEW	Vista principal de una página contenedora de PortalBeans	view.jsp
USER_BEAN_EDIT_VIEW	Configuración por parte del usuario. Se accede pulsando el botón EDITAR del Chrome del PortalBean	edit.jsp
USER_BEAN_EDIT_PROCESS_VIEW	Procesa opciones de configuración del usuario y redirige a la vista correspondiente	process_edit.jsp
ADMIN_BEAN_EDIT_VIEW	Configuración por parte del administrador. Se accede pulsando la opción de DETALLES del PortalBean desde la consola del servidor	admin_edit.jsp
ADMIN_BEAN_EDIT_PROCESS_VIEW	Procesa opciones de configuración del administrador y redirige a la vista correspondiente	process_admin_edit.jsp
GENERIC_VIEW	Muestra PortalBeans en una página que incluye la cabecera, pie y navegación genéricos del site.	
GENERIC_PROCESS_VIEW	Procesa GENERIC_VIEW y redirige a la vista correspondiente	
RESULT_SET_VIEW	Idéntica a GENERIC_VIEW. Pensada para mostrar páginas de resultados de una búsqueda	results.jsp
RESULT_PROCESS_VIEW	Idéntica a GENERIC_ PROCESS _VIEW. Procesa RESULT_SET_VIEW y redirige a la vista correspondiente	results_process.jsp
FRAME_VIEW	Muestra los contenidos de una URL en un *frame*	
REDIRECT_VIEW	Muestra el contenido mostrado por el PortalBean en una página que no incluye la cabecera, pie y navegación predefinidos del site	
RAW_VIEW	Idéntica a REDIRECT_VIEW	

En las vistas en las que se ha especificado, la ID de la vista es puede ser mapeada automáticamente contra la JSP especificada a no ser que se indique otra cosa al sobrecargar el método getView() de la clase principal (la

que extiende JSPBean). Para el resto de vistas, debe implementarse en el getView() el mapeo de ID´s de vista contra la correspondiente JSP.

Las vistas no estándar se definirían en el .pbd de la siguiente manera:

```
<VIEW ID="unaVistaNoEstandar"
        USERLEVEL="registered_user"
        IS_NAVIGATION_ROOT="true"
        PREFERRED_PAGE_ID="generic">
</VIEW>
```

Ejemplo:

```
import com.epicentric.common.Log;
import com.epicentric.portalbeans.PortalPageContext;
import com.epicentric.portalbeans.GenericBeanView;
import java.io.IOException;

public class UnaView extends GenericBeanView
{
        public UnaView (UserInfoBean bean,PortalPageContext ppc)
        {
                //Usamos el constructor de la superclase
                super(bean, ppc);
        }

        public void pageStart()
        {
                //Lógica y procesos que la vista debe efectuar al ser llamada
        }

        //Se sobrecarga el método serviceHTML() que sera invocado por el container a la hora de servir
        //la página
        public void serviceHTML()
        {
                try
                {
                        //Se redirige ala vista principal
                        getPortalPageContext().sendRedirect(getBean().getFullViewURL("MY_PORTAL_VIEW"));
                }
                catch (IOException ioe)
                {
                        Log.message(Log.kDebug, ioe, "problem in serviceHtml()");
                }
        }

        private String otroMétodo()
        {
                //Lógica adicional que la vista necesita
        }
}
```

Páginas JSP: las vistas en última instancia se muestran mediante páginas JSP. Las páginas JSP usan un objeto JSPView. Ejemplo:

```
//Las siguientes directivas son necesarias al comienzo
<%@ page contentType="text/html; charset=UTF-8" %>
<%@taglib uri="PortalBean-tags" prefix="mod" %>

<mod:view class=" com.mycompany.portalbeans.unaview.UnaView">

<%
//Con este método obtenemos una URL a la que redireccionar
        String actionURL=view.getBean().getFullViewURL(PortalBeanView.ADMIN_BEAN_EDIT_PROCESS_VIEW);
%>
```

```
        <form name="saveUserNumber" method="POST" action="<%= actionUrl %>" >
        <table border="0" class="epi-datatable">
                <tr>
                        <td>
                //Podemos invocar métodos existents en la clase JSPView
                                <input type="text" value="<%=otroMetodo%>" name="unaEntrada" />
                        </td>
                </tr>
        </table>
</form>
</mod:view>
```

Component.xml: se trata de un fichero que todo component que se quiera desplegar en Portal debe llevar en su empaquetamiento para poder ser desplegado en Portal. Debe estar situado en el raíz del .CAR en que el que se empaquetará el PortalBean.

Este fichero proporciona datos como el nombre totalmente cualificado de la clase, título, versión, descripción, ruta donde los ficheros van a ser desplegados, dependencias de otros componentes,etc.

A continuación se muestra un ejemplo.

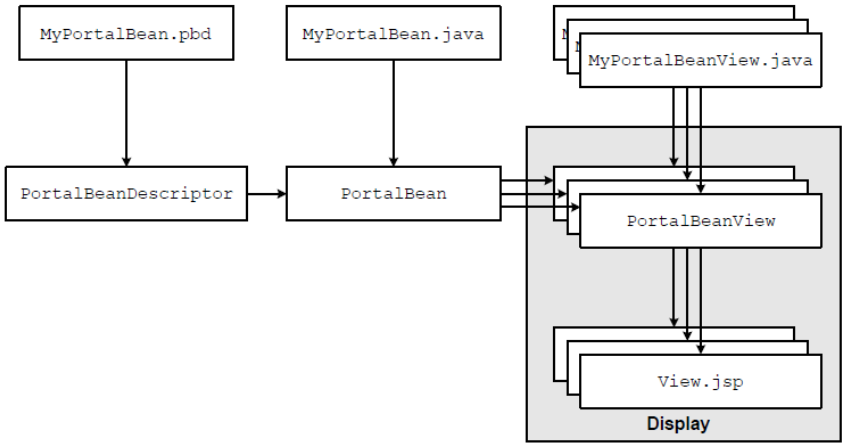
```
<?xml version="1.0" encoding="UTF-8" ?>
<epideploy:component
xmlns:epideploy="http://www.epicentric.com/deployment"
component-type="ModuleTypes"
component-id="string"
major-version="integer"
minor-version="integer"
build-version="string"
epi-version="string"
epi-build="integer"
title="free-form text"
description="string" >

        <epideploy:required-component component-type="string"
        component-id="string"
        major-version="integer"
        minor-version="integer" />

                <epideploy:detail>
                <module-config auto-deploy="true"|"false">
                        <descriptor-id>
                                Nombre del PortalBean
                        </descriptor-id>
                        <web-deployment-path>
                                file_system_directory
                        </web-deployment-path>
                        <locale-key>
                                unique_ID
                        </locale-key>
                </module-config>
        </epideploy:detail>
</epideploy:component>
```

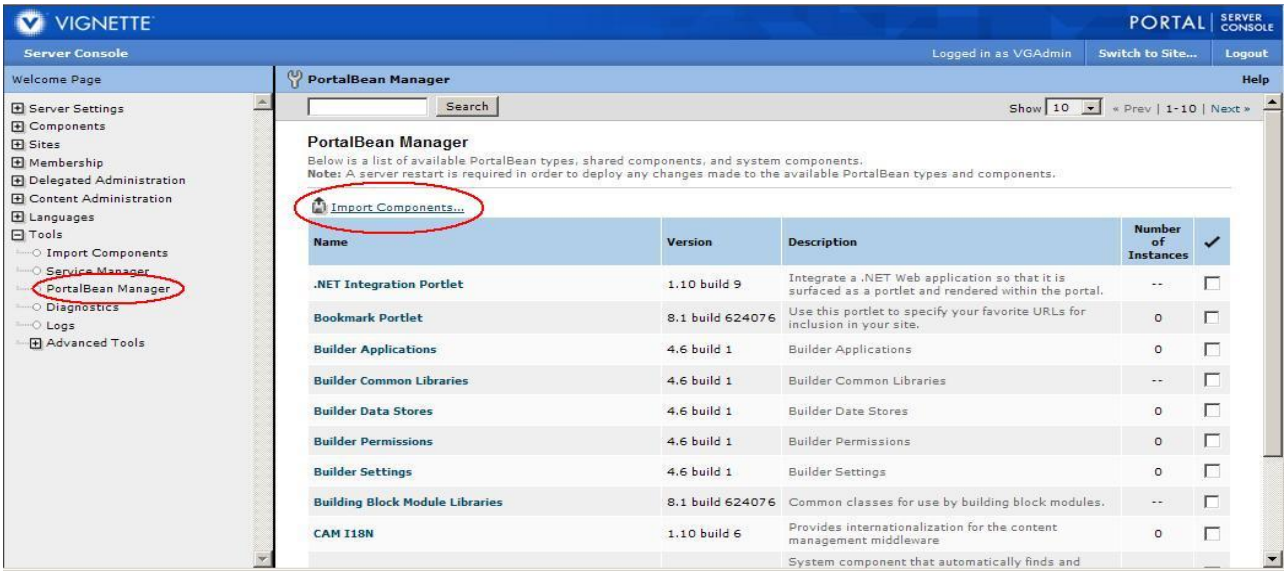
Ficheros secundarios: imágenes y otros ficheros que el PortalBean pudiere necesitar para mostrar contenidos al usuario.

A continuación se muestra la relación que existe entre los diversos ficheros/clases y sus objetos asociados

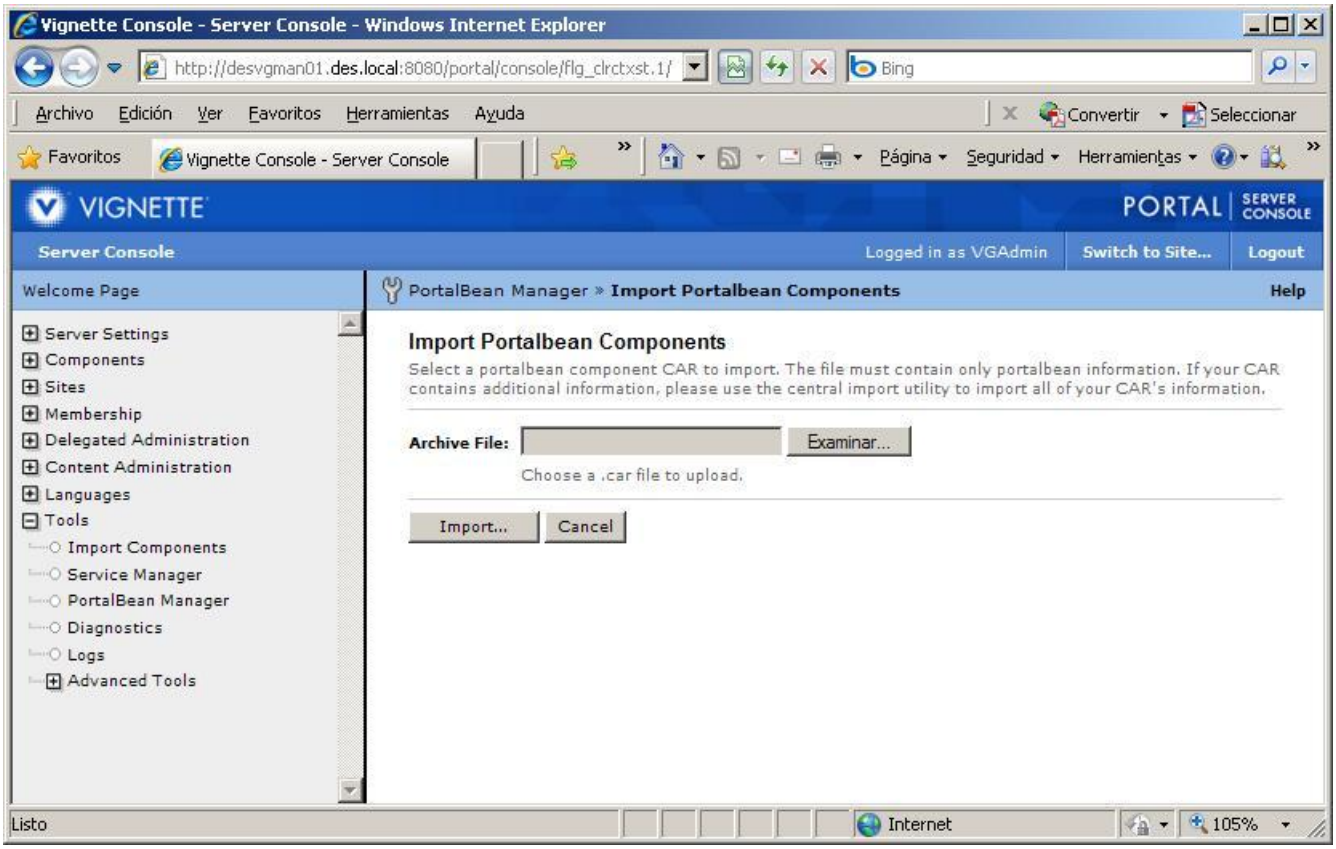


8.2.2 Despliegue de un PortalBean en Vignette Portal

Para desplegar un PortalBean en Portal hemos de crear un fichero .CAR con los contenidos del Portlet siguiendo la estructura de directorios especificada. Una vez lo tengamos, nos dirigimos, dentro de la Server console en Portal a "`Tools > PortalBean Manager`". Aparecerá un listado de los Portalbeans desplegados. Se pulsa en "`importComponents`".

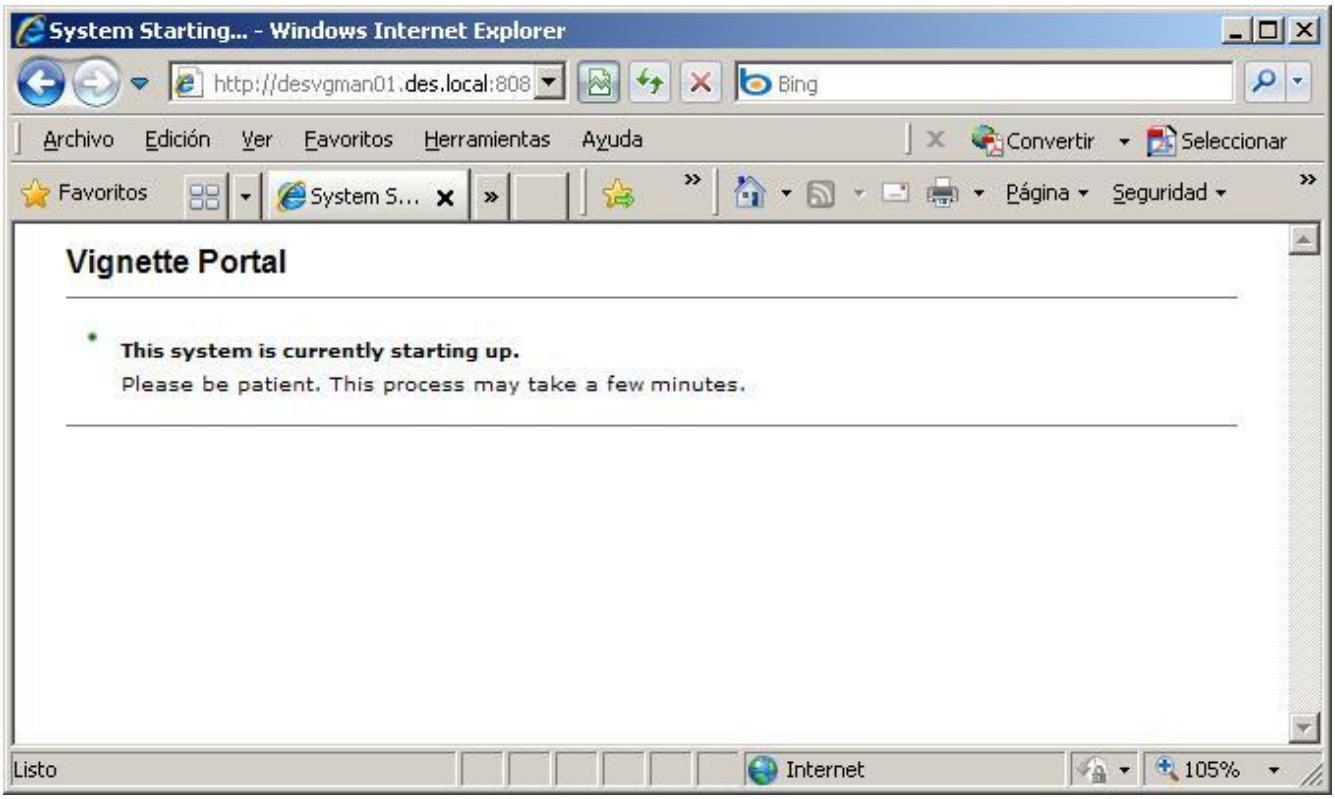


Nos llevará a un menú donde podernos introducir la ubicación del .CAR y desplegarlo. Pulsar *Import.*



Una vez desplegado es necesario detener Vignette Portal y volver a arrancarlo. Para ello accedemos a la administración de Tomcat y en ella detenemos la aplicación web *portal* y la volvemos a arrancar.

Portal tarda unos instantes en arrancar. Si en el momento en que está arrancando intentamos acceder, veremos la siguiente patalla.


System Starting... - Windows Internet Explorer
http://desvgman01.des.local:808
Bing
Archivo Edición Ver Favoritos Herramientas Ayuda
Convertir
Seleccionar
Favoritos
System S...
Página
Seguridad
Vignette Portal
This system is currently starting up.
Please be patient. This process may take a few minutes.
Listo
Internet
105%

9 Gestión de Versiones

9.1 Herramienta de Gestión de Versiones

Se propone el uso de un sistema de gestión de la configuración que permite automatizar las tareas de guardar, recuperar, registrar e integrar versiones de archivos.

Concretamente una herramienta de este estilo debe permitir:

- Consultar el histórico de modificaciones y recuperar una versión anterior del fichero.
- Comparar distintas versiones.
- Permite el bloqueo de archivos para evitar que más de una persona los esté editando a la vez.
- Permite subir listas de cambios que incluyen modificaciones en varios ficheros y tratar las como una modificación única en el código.
- Permite mantener varias ramas de versiones del código.

En el momento de creación de este documento el Instituto de Empresa dispone de dos herramientas para la realizar la gestión de la versiónes: Soursafe y TFS (Team Foundation Server).

El objetivo común tanto de Accenture como del Instituto de Empresa es disponer una herramienta que permita realizar las tareas indicadas con la mayor eficiencia posible, de forma que se pueda tener un mayor control del código desarrollado y desplegado en las diferentes máquinas y entornos.

Desde Accenture se valora apoya la iniciativa del Instituto de Empresa de migrar la versión actual del TFS a la última disponible (2010), con el fin de disponer de esta herramienta durante el desarrollo de los diferentes portales.

9.2 Estrategia de branching

A continuación se detalla el modelo de branching.

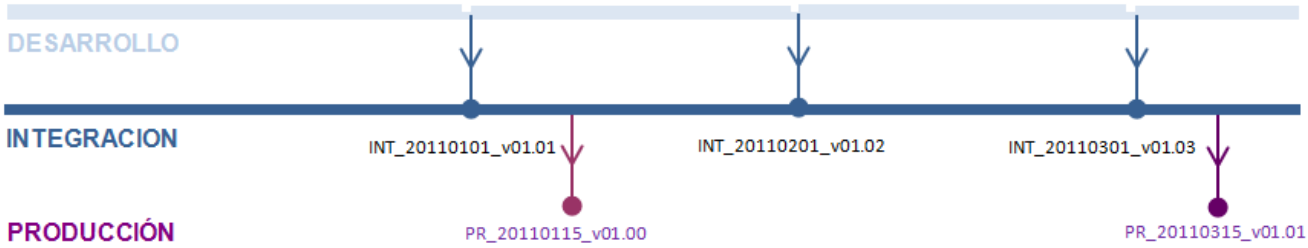
Se plantean crear tres ramas independientes, una para cada entorno: “desarrollo”, “integración” y “producción”

La rama de “desarrollo” como su propio nombre indica, es la que mayor actividad tendrá y por lo tanto menor estabilidad.

La rama de “integración” recibirá los desarrollos después de haber pasado por la fase de pruebas del entorno anterior. Será por tanto, una rama con menor actividad que la anterior y mayor estabilidad.

La rama de “producción” almacenará únicamente el contenido liberado. Será la rama con menor actividad y con mayor estabilidad, puesto que el código que llegue a esta rama, habrá pasado por dos entornos anteriormente.

La siguiente figura ilustra la propuesta de estrategia en ramas para una situación de desarrollo sin errores.



Transición entre ramas:

"desarrollo" → "integración". Tal y como se ha comentado antes, será necesario haber completado un conjunto mínimo de pruebas en "desarrollo" antes de entregar la versión a "integración"

"integración" → "producción". Este paso sólo se podrá realizar con la conformidad del usuario.

"desarrollo" → "producción". Este paso no se debería llevarse a cabo en ningún caso.

9.3 Etiquetado de componentes

Cada vez que un desarrollador suba una versión de un componente (javascript, jsp, css, …) tendrá que etiquetarlo de acuerdo al siguiente estándar:

<**aaaammdd_AAA_des**> siendo

aaaa	Año de la versión
mm	Mes de la versión
dd	Día de la versión
AAA	Siglas del desarrollador que realiza el cambio
des	Descripción de la modificación realizada (*)

(*) En caso de Este último metadato podrá ir en los comentarios asociados al tag

9.4 Etiquetado de las versiones de la aplicación

En determinadas circunstancias, dependiendo del entorno, se crearán versiones de la aplicación, que incluirá todos los componentes necesarios para la ejecución de la misma.

En entorno de desarrollo, esto se hará periódicamente, como mínimo una vez cada quince días y para ello será necesario estabilizar entre los desarrolladores el código con el fin de conseguir una versión ejecutable.

En entorno integrado, las versiones se harán cada vez que se realice un pase desde el entorno anterior.

En producción, las versiones se harán cada vez que se realice un pase desde el entorno anterior.

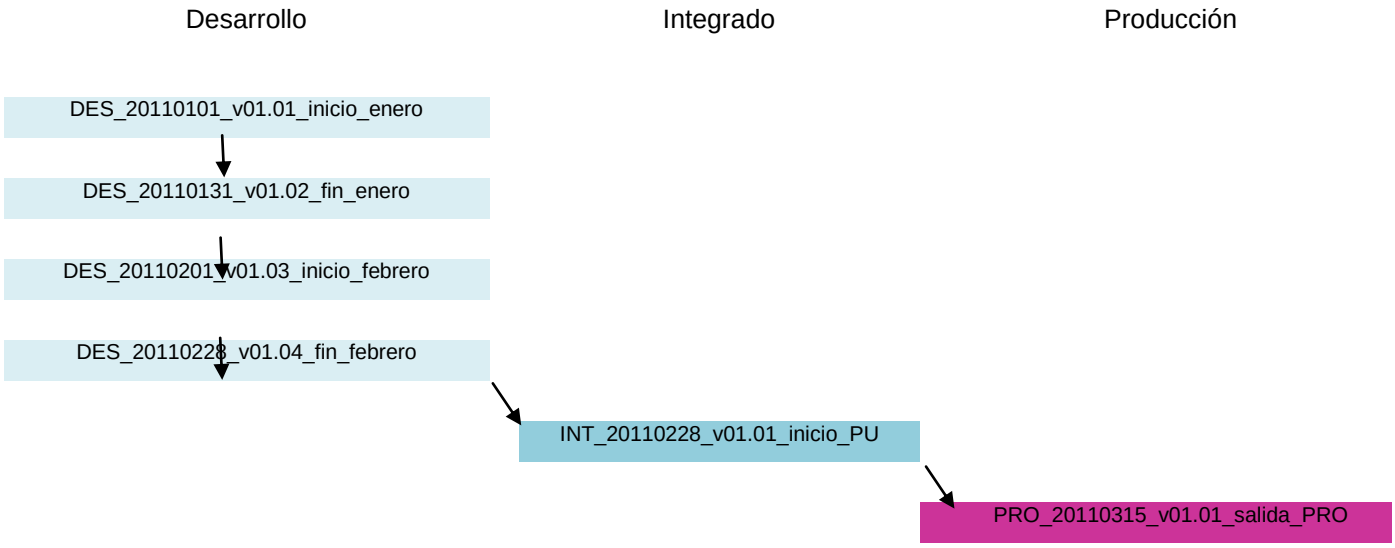
Las distintas versiones del aplicativo, también deberán cumplir un acuerdo en la nomenclatura a seguir, que será la siguiente:

<**ent_aaaammdd_vMM.mm_des**> siendo

ent	entorno para el que se crea la versión [desarrollo == DES, integrado == INT o producción == PRO]

aaaa	Año de la versión
mm	Mes de la versión
dd	Día de la versión
MM	Mayor versión, que corresponde con cada una de las versiones planificadas por el Instituto de Empresa
mm	minor versión, que corresponde con versiones menores, realizadas para solucionar fixes,….
des	Descripción de la versión

A continuación se incluye un gráfico ilustrativo del etiquetado de las versiones de la aplicación:



10 Deploy

10.1 Herramienta de despliegue: Ant

A continuación se detalla la herramienta y procedimiento para realizar el deploy.

La herramienta propuesta es: “Ant”

Ant es una herramienta Open Source, desarrollada en lenguaje Java, que no depende de los comandos Shell, lo que la hace independiente del entorno.

Los comandos ejecutados por Ant se establecen en un fichero xml, denominado “built.xml”. A través de este archivo se indican las ordenes que se desean realizar, ya sea crear directorios, realizar un ftp o compilar un proyecto.

La propuesta de despliegue supone tener instalado Ant tanto en el entorno de desarrollo como en el de integración.

Se definirá un fichero “build.xml” que permitirá en la medida de lo posible realizar la transición entre entornos. Tal y como aparece en el apartado: “Entorno local”, en la estructura de los diferentes proyectos cuando se requiere incluir algún tipo de configuración, se ha creado un subdirectorio denominado “entorno” donde se almacene los ficheros con información dependiente del entorno.

Estos ficheros no podrán ser desplegados de forma automática a través de Ant, puesto que requieren de la intervención manual para su parametrización en función del entorno.

10.2 Procedimiento de despliegue

En este capítulo se procederá a explicar cómo se realiza el despliegue de las diferentes partes de código desde el entorno local hasta producción.

El paso más complejo radica en desplegar los proyectos ubicados en Eclipse de acuerdo a la topología Vignette establecida en el entorno de desarrollo. El resto de promociones: desarrollo a entorno integrado (o preproducción) y entorno integrado a producción, serán más sencilla al compartir las máquinas origen y destino la misma arquitectura (una máquina de management en desarrollo, una máquina de management en entorno integrado, una máquina de delivery en desarrollo, una máquina de delivery en entorno integrado,…)

10.2.1 Deploy de local a desarrollo

Se van a distinguir dos tipos de componentes a subir al entorno de desarrollo desde local. En el primer apartado se va a ver cómo promocionar los proyectos que están alojados en Eclipse (que serán principalmente dinámicos: clases java, jsps,…) y a continuación los elementos estáticos (imágenes, pdf, …)

10.2.1.1 Código Eclipse

Los diferentes proyectos en local (Eclipse) tienen que desplegarse en desarrollo de acuerdo a las siguientes figuras:

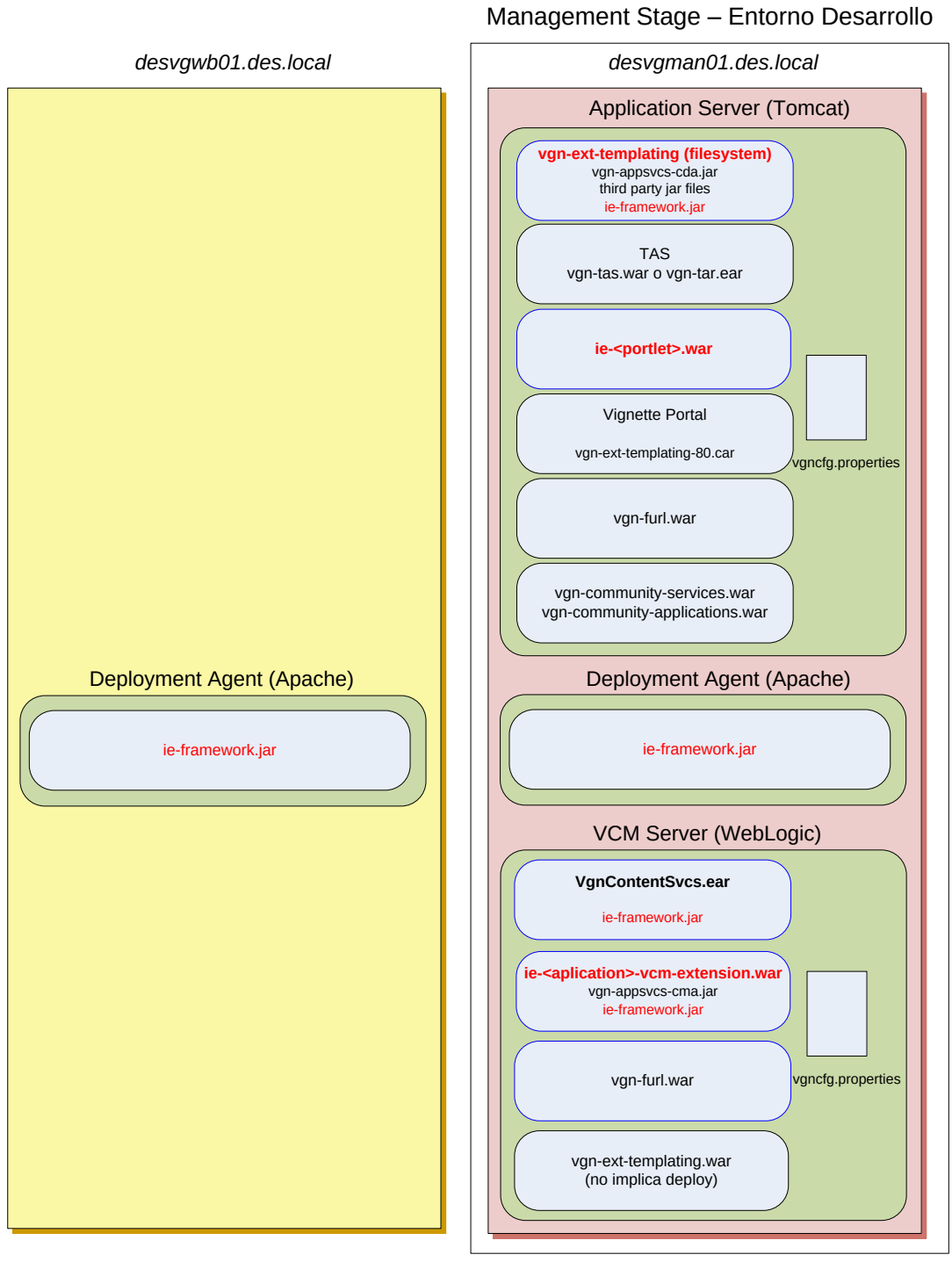


Figura ilustrativa del despliegue de los diferentes proyectos en el stage de management

Nota: En rojo aparecen los elementos que han sido desarrollados a medida, mientras que en negro aparece el software facilitado por Vignette. “vgn-ext-templating” aparece en rojo a pesar de ser código facilitado por Vignette porque también incluye desarrollos propios. Este código se encuentra desplegado en el filesystem y no como un war

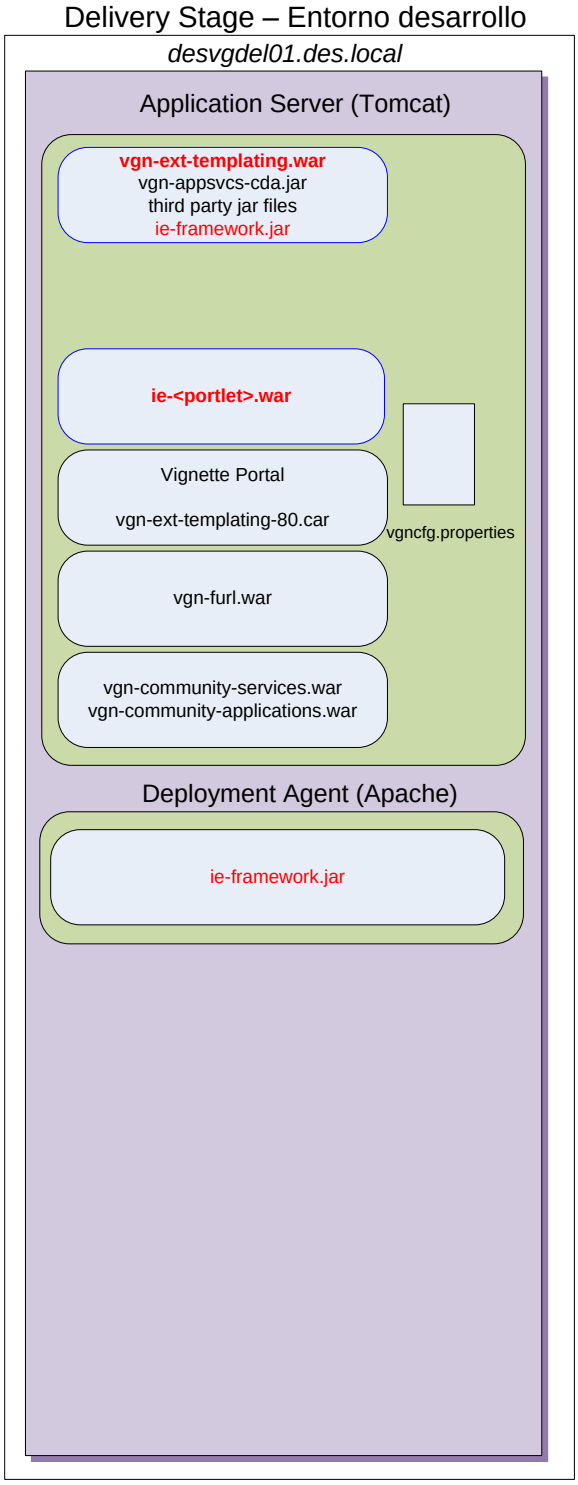


Figura ilustrativa del despliegue de los diferentes proyectos en el stage de management

Nota: En rojo aparecen los elementos que han sido desarrollados a medida, mientras que en negro aparece el software facilitado por Vignette. “vgn-ext-templating” aparece en rojo a pesar de ser código facilitado por Vignette porque también incluye desarrollos propios. Este código se encuentra desplegado en el filesystem y no como un war.

En las figuras anteriores se puede ver cómo algunos proyectos deben empaquetarse como un jar, mientras que otros deben hacerlo como un war. Además aparece la distribución de esos paquetes en las diferentes máquinas de desarrollo.

Si hubiese, por ejemplo, que desarrollar en Vignette un portal para el campus de Segovia del Instituto de Empresa y suponiendo que en Eclipse hubiese los siguientes cinco proyectos:

- **ie-framework**
- **ie-segovia-vcm-extension**
- **vgn-ext-templating**
- **ie-<portlet>**
- **ie-librerias-comunes**

El procedimiento a seguir sería:

- empaquetar el proyecto ie-framework en un jar: **ie-framework.jar**
- empaquetar el proyecto ie-segovia-vcm-extension en un war: **ie-segovia-vcm-extension.war** , sin olvidar incluir en su interior el jar: ie-framework.jar
  Para más información acerca del deploy y registro de los widget se puede consultar el documento oficial de Vignette: "Vignette Content Extensibility SDK Guide 8.0 SP1.pdf" en los apartados:
  "5 Content Extensibility > Extending Widgets > Deploying the Widget" y
  "5 Content Extensibility > Extending Widgets > Registering the Widget"
- con los cambios realizados en el **vgn-ext-templating**, actualizar el proyecto del mismo nombre que se encuentra en el tomcat (desplegado en el filesystem), sin olvidar incluir el jar: ie-framework.jar
- empaquetar el proyecto ie-<portlet>, en un war: **ie-<portlet>.war**
- empaquetar las librerías comunes de desarrollo en un jar: **ie-librerias-comunes.jar**

Con los cinco proyectos empaquetados, en dos jar, dos war y un despliegue en el filesystem , lo único que faltaría sería distribuir estos ficheros en desarrollo, de la siguiente manera:

- **ie-framework.jar**, habría que alojarlo en la máquina en el deploy agent de management, y en el ear: VgnContentSvcs.ear.
- **ie-segovia-vcm-extension.war**, se alojará en el servidor WebLogic de management
- **vgn-ext-templating,** se distribuiría a la máquina de management (Tomcat) desplegado en el filesystem y no como un war
- con **ie-<portlet>.war,** se desplegará en el Tomcat de management
- **ie-librerias-comunes.jar** se subirá al directorio de librerías comunes del servidor de aplicaciones.

A continuación se puede ver un resumen del empaquetado y distribución de los diferentes proyectos en las diferentes máquinas.

Management State > desvgwb01.des.local

Componente	Ubicación / Deploy	Proveedor	Proyecto Eclipse asociado
ie-framework.jar	Deployment Agent (Apache)	IE	ie-framework

Management State > desvgman01.des.local

Componente	Ubicación / Deploy	Proveedor	Proyecto Eclipse asociado
vgn-ext-templating.war	Application Server	Vignette	
vgn-appsvcs-cda.jar	vgn-ext-templating.war	Vignette	
third party jar files	vgn-ext-templating.war	Vignette	
ie-framework.jar	vgn-ext-templating.war	IE	ie-framework
TAS (vgn-tas.war o vgn-tar.ear)	Application Server	Vignette	

ie-<portlet>.war	Application Server	IE	ie-<portlet>
VAP (vgn-ext-templating-80.car)	Application Server	Vignette	
vgn-furl.war	Application Server	Vignette	
vgn-community.war **vgn-community-applications.war**	Application Server	Vignette	
ie-framework.jar	Deployment Agent (Apache)	IE	ie-framework
VgnContentSvcs.ear	VCM Server	Vignette	
ie-framework.jar	VgnContentSvcs.ear	IE	ie-framework
ie-<application>-vcm-extension.war	VCM Server	IE	ie-<application>-vcm-extension
vgn-appsvcs-cma.jar	ie-<application>-vcm-extension.war	Vignette	
ie-framework.jar	ie-<application>-vcm-extension.war	IE	ie-framework
vgn-furl.war	VCM Server	Vignette	
vgn-ext-templating.war	VCM Server	Vignette	
vgncfg.properties	VCM Server	Vignette	

Delivery State > desvgdel01.des.local

Componente	Ubicación / Deploy	Proveedor	Proyecto Eclipse asociado
vgn-ext-templating.war	Application Server	Vignette	
vgn-appsvcs-cda.jar	vgn-ext-templating.war	Vignette	
third party jar files	vgn-ext-templating.war	Vignette	
ie-framework.jar	vgn-ext-templating.war	IE	ie-framework
ie-<portlet>.war	Application Server	IE	ie-<portlet>
VAP (vgn-ext-templating-80.car)	Application Server	Vignette	
vgn-furl.war	Application Server	Vignette	
vgn-community.war **vgn-community-applications.war**	Application Server	Vignette	
ie-framework.jar	Deployment Agent (Apache)	IE	ie-framework
ie-framework.jar	ie-<application>-vcm-extension.war	IE	ie-framework
vgn-furl.war	VCM Server	Vignette	
vgn-ext-templating.war	VCM Server	Vignette	

vgncfg.properties	VCM Server	Vignette	

10.2.1.2 Componentes estáticos

En este punto se describe el pase de las modificaciones de los ficheros estáticos (css, js, imágenes,…)

Pendite aclarar tras consultar al VPS

10.2.2 Deploy en el resto de entornos

Al igual que se hizo en "10.2.1De local a desarrollo", en este apartado se va a explicar la promoción entre entornos de los componentes en función del tipo de los mismos.

10.2.2.1 Código Eclipse

Una vez que la aplicación este funcionando y estabilizada en el entorno de desarrollo, con el reparto de componentes indicados anteriormente, se procederá a realizar su subida a los diferentes entornos, utilizando siempre que sea posible Ant, que permitirá subir los componentes y desplegarlos en el entorno destino. Según el desarrollo realizado, la versión podrá requerir ciertas acciones manuales, como son cambiar configuración dependiente del entorno.

10.2.2.2 Código Vignette

El proceso de despliegue entre entornos de los contenidos de Vignette incluye dos puntos fundamentales, el:

1. Deploy (export – import) de canales, contenidos y configuración de dynamic

1.1. Exportación de los canales VCM modificados o nuevos, a fichero (pkg1.zip**)
Para detallar suficiente esta tarea, sin perder en enfoque global, se ha optado por mover esta información al Apéndice A de este documento.

1.2. Exportación de las páginas asociadas a los canales modificados, a un fichero (pkg2.zip**)
Ver detalle de esta tarea en el Apéndice B de este documento

1.3. Exportación de los canales VCM con los contenidos asociados a fichero (pkg3.zip **)
Ver detalle de esta tarea en el Apéndice C de este documento

1.4. Despublicar los canales afectados en el nuevo entorno.

1.5. Mover los ficheros exportados anteriormente a una ruta del filesystem del nuevo entorno

1.6. Importar, utilizando vgnimport (/opt/Vignette/VCM/Content/8_0/bin/vgnimport) los ficheros en el orden correcto.

```
vgnimport -c conflictpolicy=overwrite -u [user] -p [password] –h [host]:[port]
-f [ruta/pkg1.zip]
vgnimport -c conflictpolicy=overwrite -u [user] -p [password] –h [host]:[port]
-f [ruta/pkg2.zip]
vgnimport -c conflictpolicy=overwrite -u [user] -p [password] –h [host]:[port]
-f [ruta/pkg3.zip]
```

-u *VGAdmin*	Usuario de acceso a VCM
-p <password>	Password de acceso a VCM

-h *desvgwb01.des.local:27110*	Maquina y puerto para acceso al VCM
-f *<fichero>.zip*	Nombre del zip a importar

1.7. Acceder al Gestor de Contenidos VCM y navegar hasta la ruta de los Channels afectados

1.8. Publicar los canales importados en el nuevo entorno.

Nota **: El nombre indicado de los ficheros de exportación no es obligatorio.

Nota: La importación de los ficheros debe hacerse antes de hacer la configuración del Site.

2. Deploy (export- import) de un Site

2.1. Exportar el Site

2.2. Importar el Site

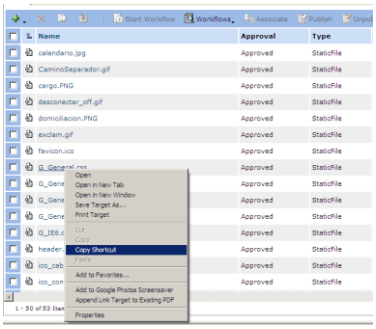
Este proceso permite migrar los cambios realizados en la Consola de Administración de VAP, que incluye modificaciones en grids, styles, paginas y configuración de portlets. Para el detalle de las tareas de exportación e importación de un Site ver Apéndice D de este documento.

10.2.2.3 Componentes estáticos

Los paso a seguir para realizar la promoción de los contenidos estáticos entre entornos son:

1.- Obtener el identificador de Vignette de los contenidos modificados

Identificar el contenido en Vignette Content Manager. Con botón derecho sobre el link del contenido seleccionar la opción “Copiar Acceso Directo” o “Copy Shortcut”.



Se copiará la siguiente instrucción en el portapapeles:

```
javascript:assetRedirect  (
   document.VignConsoleForm,
   '434b48c806573210VgnVCM1000004316700aSTFL',
   '/AppConsole/secure/properties.do'
)
```

El segundo parámetro es el identificador de Vignette del contenido seleccionado.

2.- Exportar contenidos estáticos VCM (VgnExport)

Este procedimiento permite exportar StaticFiels subidos a Vignette Content Management de un entorno a otro, exportando también el identificador de Vignette asignado al contenido estático y poder importarlo a otro entorno manteniendo los identificadores de Vignette del fichero. Para ello hay que:

2.1.- Acceder por consola de comandos a la maquina donde está instalado el VCM.
2.2.-Crear un nuevo directorio para guardar los exports a generar:

```
mkdir /home/vignette/Vignette/vgnexport_aaaammdd/
```

2.3.-Navegar hasta el directorio creado en el paso anterior:

```
cd /home/vignette/Vignette/vgnexport_aaaammdd/
```

2.4.- Ejecutar el siguiente commando:

```
/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh -u VGAdmin -h desvgwb01.des.local:27110
-i 889248c806573210VgnVCM1000004316700aSTFL -f utilidades_js.zip
```

/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh	Comando a Ejecutar. El directorio "/opt/Vignette/VCM/" se corresponde con el directorio de instalación del VCM.
-u *VGAdmin*	Usuario de acceso a VCM
-h *desvgwb01.des.local:27110*	Maquina y puerto para acceso al VCM
-i *889248c806573210VgnVCM1000004316700aSTFL*	Identificador del fichero estático que se quiere exportar. Ver punto 1("obtener el identificador de Vignette de los contenidos modificados")
-f *utilidades_js.zip*	Nombre del zip a generar con la exportación. Se recomienda: nombreFicheroEstatico_extension.zip

3.- Importar contenidos estáticos VCM (VgnImport)

El siguiente paso consistirá en importar los ficheros estáticos al nuevo entorno. La herramienta de Vignette que se utiliza con estos fines es vgnimport. Estos ejecutables se encuentran en la siguiente ruta: `/opt/Vignette/VCM/Content/8_0/bin/`

Copiar los ficheros entregados en un directorio temporal de la máquina donde está instalado VCM.

Ejecutar la siguiente sentencia para cada fichero entregado:

```
vgnimport -c errorpolicy=continue -u [user] -p [password] -h [host]:[port] -f [ruta fichero.zip]
```

-u *VGAdmin*	Usuario de acceso a VCM
-p <password>	Password de acceso a VCM
-h *desvgwb01.des.local:27110*	Maquina y puerto para acceso al VCM
-f *<fichero>.zip*	Nombre del zip a importar

4.- Publicar contenidos estáticos VCM

Acceder al Gestor de Contenidos VCM, navegar hasta la carpeta Content/branding y publicar los contenidos importados en el paso anterior.

Documentación de referencia

- Vignette Dynamic Portal Module and Vignette Dynamic Site Module 8.0 Developers Guide.pdf
- Vignette Content Extensibility SDK Guide 8.0 SP1.pdf
- http://java.sun.com/docs/codeconv/CodeConventions.pdf
- IE – Guia para el desarrollador_Nomenclatura Java
- IE – Documento de Arquitectura.docx

Apéndice A – Exportación de los canales VCM modificados o nuevos

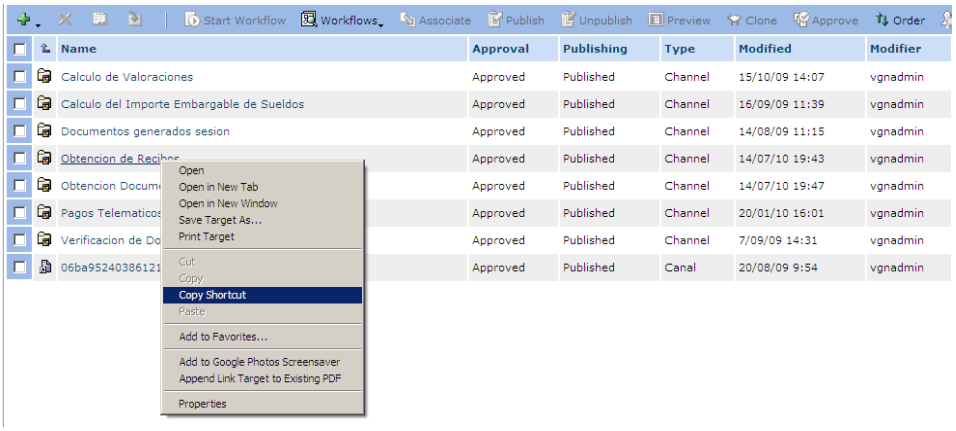
En este apéndice describe el proceso de exportación de los canales de VCM cuando habiendo creado o modificado un canal de VCM (se han cambiado o añadido contenidos y/o elementos de presentación) se desea migrar esta modificación al siguiente entorno de desarrollo.

A continuación se detalla los pasos necesarios para realizar la exportación de los canales:

1. Identificar los Channels en VCM.

Con botón derecho sobre el link al Channel seleccionar la opción “Copiar Acceso Directo” o “Copy Shortcut”.

Se copiará la siguiente instrucción en el portapapeles:




```
javascript:containerRedirect(

document.VignConsoleForm,
'/de981d7db4681210VgnVCM1
00000ca0a700aPROJ/1448952
403861210VgnVCM1000000100
007fRCRD/1e38952403861210
VgnVCM1000000100007fRCRD/
8758952403861210VgnVCM100
0000100007fRCRD/ec8a95240
3861210VgnVCM100000010000
7fRCRD/06ba952403861210Vg
nVCM1000000100007fRCRD/dc
bb952403861210VgnVCM1000000100007fRCRD','/Sites/…/… ')
```

El texto del segundo parámetro marcado es el identificador de Vignette del canal.

2. Generar el fichero manifest con los identificadores de los canales

Se tendrá que generar un fichero manifest con los identificadores de los Channels obtenidos, que tenga la siguiente estructura (chManifest.xml):

```
<?xml version="1.0" encoding="UTF-8" ?>
<packageManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.vignette.com/xmlschemas/importexport"
xsi:schemaLocation="http://www.vignette.com/xmlschemas/importexport
7202/packageManifest.xsd">
<imports>
       <object id=" dcbb952403861210VgnVCM1000000100007fRCRD"/>
</imports>
</packageManifest>
```

3. Exportación de los canales

3.1.- Acceder por consola de comandos a la maquina donde está instalado VCM.

3.2.- Crear un nuevo directorio para guardar los exports a generar:

```
mkdir /home/vignette/Vignette/vgnexport_aaaammdd/
```

3.3.- Navegar hasta el directorio creado en el paso anterior:

```
cd /home/vignette/Vignette/vgnexport_aaaammdd/
```

3.4.- Ejecutar el commando:

```
/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh -m
/home/vignette/manifest/chManifest.xml -c channelAssocs=false -u VGADmin -p … -h
desvgwb01.des.local:27110-f /home/vignette/manifest/pkg1.zip
```

/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh	Comando a Ejecutar. El directorio "/opt/Vignette/VCM/" se corresponde con el directorio de instalación del VCM.
-m */home/vignette/manifest/chManifest.xml*	Se señala la ruta donde está el fichero manifest creado en el punto anterior. (chManifest.xml)
-c *channelAssocs=false*	Se indica que no se lleve los contenidos asociados al canal.
-u *VGAdmin*	Usuario de acceso a VCM
-p <password>	Password de acceso a VCM
-h *desvgwb01.des.local:27110*	Maquina y puerto para acceso al VCM
-f */home/vignette/manifest/pkg1.zip*	Ruta donde generar el fichero con la exportación y nombre del mismo. (pkg1.zip)

El fichero generado, pkg1.zip será utilizado durante la fase de importación en el siguiente entorno.

Apéndice B – Exportación de páginas de presentación y configuración de regiones

En este apéndice describe el proceso de exportación entre entornos, cuando se han creado o modificado las Presentation Templates asociadas a los Channels del Apéndice A, o se ha modificado la configuración de sus regiones.

A continuación se detalla los pasos necesarios para realizar esta exportación:

1. Ejecutar GenSiteManifest.jsp

Ejecutar GenSiteManifest.jsp para generar el manifest de todas las Presentation Templates del portal. Acceder a la siguiente URL: http://desvgwb01.des.local:8080/vgn-ext-templating/GenSiteManifest.jsp . Introducir el nombre del site a exportar y ejecutar. Se exportará un manifest con todas las páginas del portal.

2. Creación del manifest

Editar el manifest generado para que solo queden las páginas asociadas a los Channels exportados en el paso anterior. Para cada Channel se corresponde una página. La estructura del manifest (`dpmManifest.xml`) generado debe ser la siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<packageManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.vignette.com/xmlschemas/importexport"
xsi:schemaLocation="http://www.vignette.com/xmlschemas/importexport
7202/packageManifest.xsd">
<imports>    <!-- Page for Channel – Profesional Development -->
<object id="fe5468674eeb1210VgnVCM1000001e02a8c0RCRD"/>
</imports>
</packageManifest>
```

3. Exportación de las páginas

3.1.- Acceder por consola de comandos a la maquina donde está instalado el VCM.
3.2.- Crear un nuevo directorio para guardar los exports a generar:

```
mkdir /home/vignette/Vignette/vgnexport_aaaammdd/
```

3.3.- Navegar hasta el directorio creado en el paso anterior:

```
cd /home/vignette/Vignette/vgnexport_aaaammdd/
```

3.4.- Ejecutar el siguiente commando:

```
/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh -m
/home/vignette/manifest/dpmManifest.xml -c channelAssocs=false -u VGAdmin -p … -h
desvgwb01.des.local:27110 -f /home/vignette/manifest/pkg2.zip
```

/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh	Comando a Ejecutar. El directorio "/opt/Vignette/VCM/" se corresponde con el directorio de instalación del VCM.
-m */home/vignette/manifest/dpmManifest.xml*	Indicamos la ruta donde está el fichero manifest creado en el punto anterior. (dpmManifest.xml)
-c *channelAssocs=false*	Indicamos que no se lleve los contenidos asociados al canal.
-u *VGAdmin*	Usuario de acceso a VCM
-p *<password>*	Password de acceso a VCM
-h *desvgwb01.des.local:27110*	Maquina y puerto para acceso al VCM
-f */home/vignette/manifest/pkg2.zip*	Ruta donde generar el fichero con la exportación y nombre del mismo. (pkg2.zip)

El fichero generado, pkg2.zip será utilizado durante la fase de importación en el siguiente entorno.

Apéndice C – Exportación de canales VCM con sus contenidos asociados

En este apéndice describe el proceso de exportación de los canales VCM con sus contenidos asociados para su posterior importación en el siguiente entorno de desarrollo. Este procedimiento se debe hacer si se han creado o modificado Channels en VCM.

A continuación se detalla los pasos necesarios para realizar esta exportación:

1. Ejecutar la exportación usando el manifest chManifest.xml

1.1 Acceder por consola de comandos a la maquina donde está instalado el VCM.
1.2 Crear un nuevo directorio para guardar los exports a generar:

```
mkdir /home/vignette/Vignette/vgnexport_aaaammdd/
```

1.3 Navegar hasta el directorio creado en el paso anterior:

```
cd /home/vignette/Vignette/vgnexport_aaaammdd/
```

1.4 Ejecutar el siguiente commando:

```
/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh -m
/home/vignette/manifest/chManifest.xml -c channelAssocs=true -u VGAdmin -p … -h
desvgwb01.des.local:27110 -f /home/vignette/manifest/pkg3.zip
```

/opt/Vignette/VCM/Content/8_0/bin/vgnexport.sh	Comando a Ejecutar. El directorio "/opt/Vignette/VCM/" se corresponde con el directorio de instalación del VCM.
-m */home/vignette/manifest/dpmManifest.xml*	Indicamos la ruta donde está el fichero manifest creado en el punto anterior. (dpmManifest.xml)
-c *channelAssocs=true*	Se indica que se lleve los contenidos asociados al canal.
-u *VGAdmin*	Usuario de acceso a VCM
-p *<password>*	Password de acceso a VCM
-h *desvgwb01.des.local:27110*	Maquina y puerto para acceso al VCM
-f */home/vignette/manifest/pkg3.zip*	Ruta donde generar el fichero con la exportación y nombre del mismo. (pkg3.zip)

El fichero generado, pkg3.zip será utilizado durante la fase de importación en el siguiente entorno

Apéndice D – Exportación e importación de un Site

A continuación se detalla los pasos necesarios para realizar la exportación de un Site y su posterior importación en el siguiente entorno de desarrollo

1. Exportación del Site

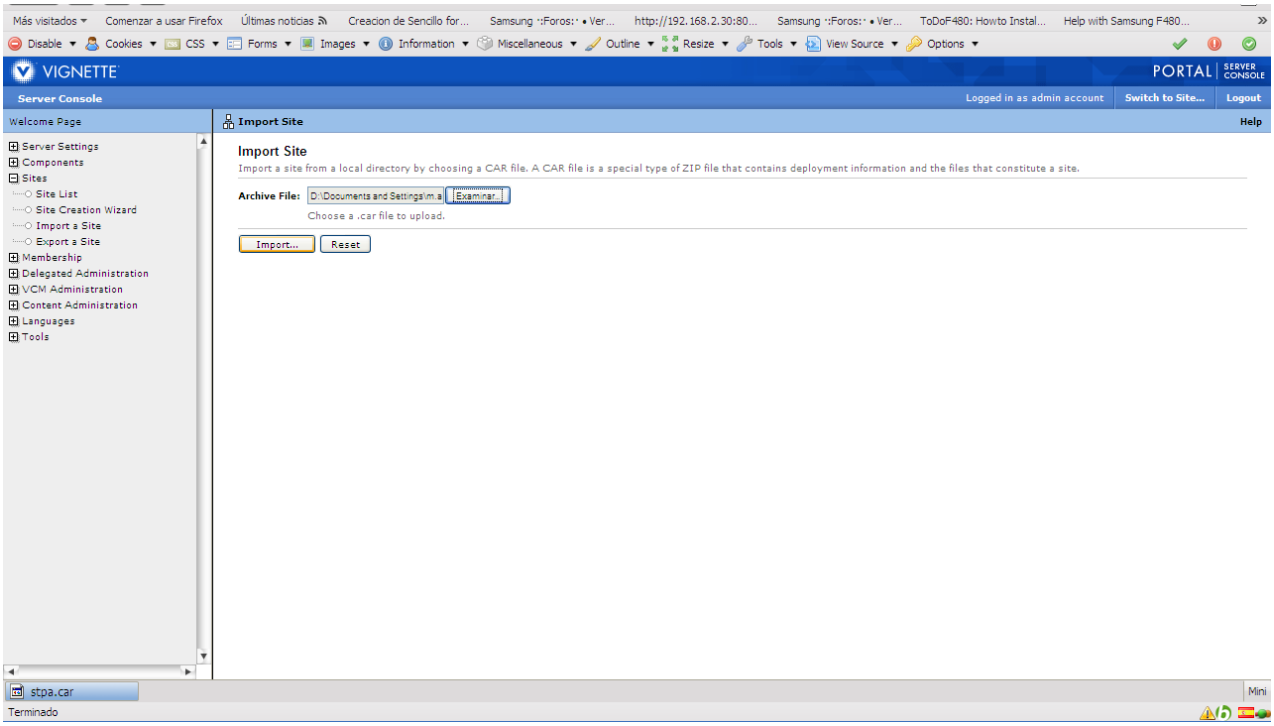
Para Importar todo lo desarrollado en Portal, a nivel de Site, se utilizará una herramienta incluida en el producto. El procedimiento es el siguiente:

- Acceder a la "*Server Console"* de Portal,
- En el menu lateral "*Sites*", seleccionar "Export Sites"
- Exportar el Site <nombreSite>, pulsando sobre el botón "*Exportsite*"
- El explorador abrirá una ventana emergente donde se debe indicar donde guardar el fichero generado con el Site. Vignette Portal utiliza una extensión propia para el tratamiento de importaciones y exportaciones de Sites, esta extensión es ".car".
- Una vez guardado el fichero generado, será posible moverlo al nuevo entorno.

2. Importación del Site

En el nuevo entorno habrá que realizar la importación de site, para ello habrá que:

- Acceder a la "*Server Console"* de Portal
- En el menú lateral "Sites", seleccionar "Import Sites"
- Automáticamente se abrirá una nueva ventana de selección en la que se puede localizar el fichero .car generado durante la exportación
- Pulsar "*Import...*" para la importación del site.



Temas pendiente

Capítulo	Punto Pendiente
Instalación Software Desarrollo	Instalación servidor local para pruebas (tomcat) para debug. Pdte VPS
Apartado: API Vignette	Pendiente crear el punto de la API de Vignette
Deploy DE elementos estáticos	Pendiente preguntar al VPS
TODOS	Revisar los pantallazos, rutas de maquinas, usuarios,…para q no aparezca nada PA