MANUAL DE USUARIO

Gestión Avanzada de ficheros de configuración

Edición: **V1.0**

Autor: **Unidad de Informática Corporativa**
Área de Desarrollo y Mantenimiento

Fecha: Mayo 2006

Historia del Documento			
Versión: **V1.1**	Manual de usuario del componente de gestión avanzada de ficheros de configuración		
	Elaborado por:	**TB-SOLUTIONS**	*Fecha*: **Diciembre 2005**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Abril 2006**
			Fecha:
			Fecha:
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Mayo 2006**

Lista de distribución del documento

Nombre	Localización

Índice

1 INTRODUCCIÓN

Este documento describe el sistema de Gestión Avanzada de Ficheros de Configuración incorporado en el framework de desarrollo de aplicaciones J2EE de la Junta de Castilla y León.

En este documento se hará un recorrido de las características básicas del componente, organizando su exposición de acuerdo a los siguientes capítulos:

- En el capítulo 2 se incluye una introducción a este sistema
- En el capítulo 3 se detallan brevemente las condiciones de entorno que requiere el componente para ser utilizado.
- En el capítulo 4 se lleva a cabo un repaso rápido y práctico de los elementos fundamentales del componente con el fin de entender fácilmente su uso.
- En el capítulo 5 se hace un recorrido detallado de las diferentes características de los elementos presentados en el capítulo anterior.

2 DESCRIPCIÓN DEL COMPONENTE

Todas las aplicaciones tienen la necesidad de almacenar parámetros operativos de forma externa al código, con el objetivo de facilitar su despliegue en entornos de producción y permitir la adecuada integración con elementos externos (servidores de correo, de ficheros, otras aplicaciones, etc.)

El framework de desarrollo para aplicaciones J2EE de la Junta de Castilla y León gestiona la configuración de una forma centralizada mediante el componente GestorConfiguración descrito en este documento. Las principales características que relacionan este componente son:

- Permite a las aplicaciones almacenar su configuración tanto en ficheros de propiedades (.properties) como en ficheros en xml.
- Permite gestionar de forma transparente múltiples ficheros de configuración. Los parámetros pueden estar en uno o varios ficheros, que pueden agruparse por contextos o tratarse de forma centralizada.
- La aplicación tiene acceso a los cambios realizados en los ficheros de configuración sin necesidad de reiniciar la ejecución (recarga en caliente).
- La memoria ocupada por las colecciones de parámetros de una aplicación se optimiza siempre al máximo, gracias a un mecanismo dinámico de carga y desalojo de contextos de configuración.

El acceso a los ficheros de configuración y la gestión de los mismos se realiza gracias a las librerías proporcionadas por el proyecto **commons-configuration** del proyecto Jakarta.

http://jakarta.apache.org/commons/configuration/

3 INSTALACIÓN DEL COMPONENTE

Este apartado describe los aspectos necesarios para incorporar el gestor de configuración a sus aplicaciones.

3.1 Librería con el componente

El gestor de configuración se encuentra empaquetado en el fichero **jcylconfiguracion-1.0.jar**. Para que la aplicación pueda hacer uso de este sistema debe copiar este fichero en una ubicación que forme parte del classpath de la máquina virtual de Java.

En el caso de aplicaciones web, puede bastar con copiarlo en el directorio WEB-INF/lib de la aplicación o en el classpath del servidor de aplicaciones.

En el caso de aplicaciones locales, puede copiarlo en el classpath de la máquina virtual de Java que ejecute la aplicación.

3.2 Librerías dependientes

Para el adecuado funcionamiento del componente, este requiere acceso a una serie de utilidades externas tomadas de otras librerías que por tanto son necesarias también dentro del proyecto de la aplicación que hace uso del componente.

El listado de librerías de las que depende el componente son las siguientes:

- commons-beanutils (versión 1.6.1)
- commons-collections (version 3.1)
- commons-configuration (version 1.1)
- commons-digester (versión 1.3)
- commons-jxpath (versión 1.2)
- commons-lang (versión 2.1)
- commons-logging (versión 1.0.3)
- commons-validator (versión 1.0.2)
- log4j (versión 1.2.7)

Al igual que la del componente, es necesario que este conjunto de librerías se encuentren en una ubicación que forme parte del classpath de la máquina virtual de Java.

4 GUÍA RÁPIDA DE USO

En este apartado se presenta un ejemplo sencillo de utilización del componente de configuración. La referencia detallada puede consultarla en el apartado siguiente.

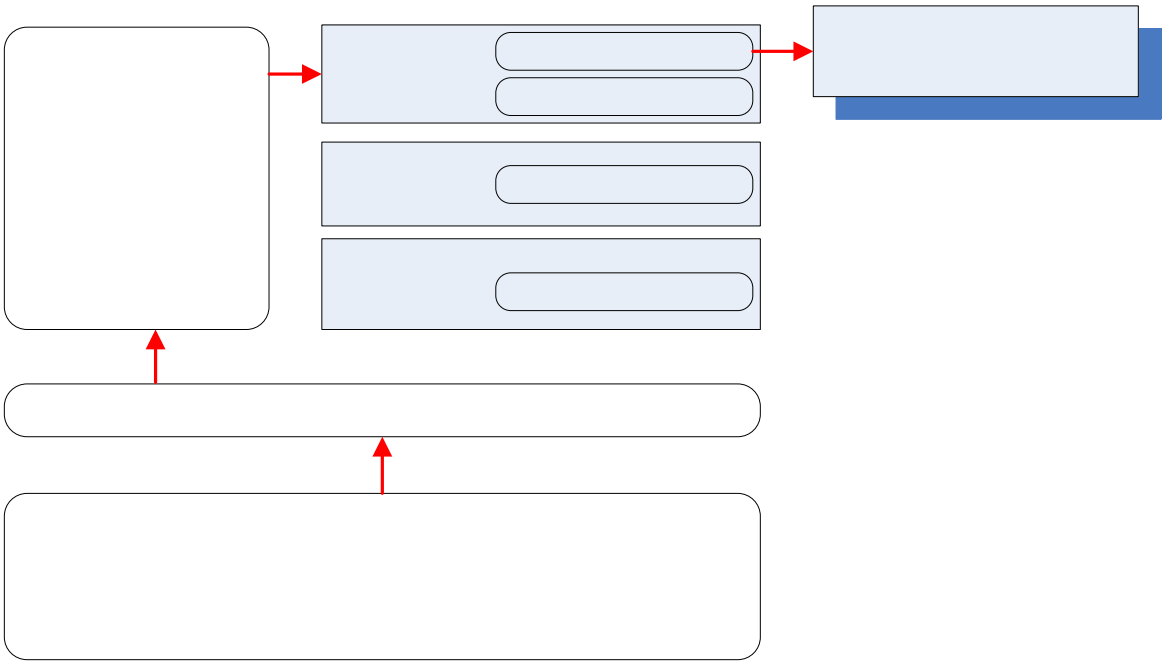
4.1 Elementos para configuración de la aplicación

Para que una aplicación haga uso del sistema de configuración, debe incorporar un fichero denominado ***repositorio de configuraciones***, que contiene la referencia a todos los ficheros de configuración de la aplicación.

Para desacoplar la aplicación de la forma de ubicar los ficheros, este repositorio define el concepto de **contexto** como una agrupación de ficheros de configuración. Cuando la aplicación desee leer un parámetro de configuración lo hará especificando el contexto en que se encuentra.

La configuración de la aplicación como tal se encuentra repartida en uno o varios **ficheros**, siempre asociados a un contexto. Este componente soporta los tradicionales ficheros .properties y ficheros XML

Esquemáticamente puede verse el conjunto en este diagrama:



Un ejemplo sencillo del fichero que define el repositorio se representa a continuación:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configurations version="1.0" xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance' xsi:noNamespaceSchemaLocation='/xsd/ConfigurationRepository.xsd'>

   <!-- Configuraciones globales -->
   <globals>
       <!-- Ruta base para todos los ficheros de configuración -->
       <initialPath value="/datos/aplicaciones/tb-solutions/configurador/"/>

       <!-- Propiedades globales -->
       <properties>
           <property name="minimumCheckDelay" value="10000"/>
           <property name="contextTimeOut" value="10000"/>
       </properties>
   </globals>

   <!-- Lista de contextos de la aplicación -->
   <contexts>
       <!-- Configuración de la aplicación -->
       <context id="default" name="Configuración de la aplicación">

           <configurationFile
                    id="application"
                    name="Parámetros generales de la aplicación"
                    location="/pruebas/config/prueba.properties"/>

       </context>

       <!-- Configuración del módulo de PDF -->
       <context id="pdf" name="Componente de generación de documentos PDF">

           <configurationFile
                    id="application"
                    name="Parámetros del componente"
                    location="./pdf.xml"/>
       </context>

   </contexts>
</configurations>
```

Este repositorio contiene los siguientes elementos:

- Una lista de propiedades globales que permiten la parametrización del propio gestor de configuraciones.

```
<!-- Configuraciones globales -->
<globals>
   <!-- Ruta base para todos los ficheros de configuración -->
   <initialPath value="/datos/aplicaciones/tb-solutions/configurador/"/>

   <!-- Propiedades globales -->
   <properties>
       <property name="minimumCheckDelay" value="10000"/>
       <property name="contextTimeOut" value="10000"/>
   </properties>
</globals>
```

- Un contexto **default** que para la configuración básica de la arquitectura de la aplicación. Contiene un único fichero de configuración basado en un fichero de propiedades estándar, denominado **prueba.properties**.

```
<!-- Configuración de la aplicación -->
<context id="default" name="Configuración de la aplicación">

    <configurationFile
         id="application"
         name="Parámetros generales de la aplicación"
         location="/pruebas/config/prueba.properties"/>
</context>
```

- Un contexto **pdf** para la configuración de un componente para generación de ficheros en formato pdf. Contiene un único fichero de configuración basado en xml denominado **pdf.xml**.

```
<!-- Configuración del módulo de PDF -->
<context id="pdf" name="Componente de generación de documentos PDF">

    <configurationFile
         id="application"
         name="Parámetros del componente"
         location="./pdf.xml"/>

</context>
```

4.2 Acceso a la configuración desde la aplicación

Para permitir que la aplicación localice el fichero con el repositorio de configuraciones debe ser inicializada indicando la ruta del fichero.

Este proceso debe realizarse una única vez durante la ejecución de la aplicación. Obsérvese que el uso del componente GestorConfiguración se realiza mediante métodos estáticos, por lo que no es necesario almacenar ninguna referencia al mismo.

4.2.1 Inicialización para aplicaciones web

En el caso de aplicaciones web, basta con añadir una nueva entrada en el fichero de despliegue web.xml de su aplicación, indicando la ruta donde se encuentra:

```
  <env-entry>
     <env-entry-name>configurationRepository</env-entry-name>
     <env-entry-value>applications/demo/demo/WEB-
INF/config/ConfigurationRepository.xml</env-entry-value>
     <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
```

Desde la aplicación debe obtener este valor y emplearlo para inicializar el componente GestorConfiguracion.

```
// Obtiene la ruta al configuration repository
String ruta = getEnvEntry("configurationRepository");

// Inicializa GestorConfiguracion a partir de la ruta donde se encuentra
GestorConfiguracion.initialize( ruta );
```

4.2.2 Inicialización para aplicaciones locales

Un posible mecanismo de inicialización para el caso de una aplicación independiente puede basarse en el uso de un parámetro de invocación. Al ejecutar la máquina virtual de Java se establece como variable de entorno la ruta del fichero:

```
        java –Druta=/config/ConfigurationRepository.xml –jar prueba.jar
```

A continuación se accede a ese parámetro para inicializar el gestor de la configuración:

```
// Obtiene la ruta al configuration repository
String ruta = System.getProperty("ruta");

// Inicializa GestorConfiguracion a partir de la ruta donde se encuentra
GestorConfiguracion.initialize( ruta );
```

4.2.3 Obtención del objeto Configuration

Una vez inicializado el componente obtenga el objeto Configuration sobre el que desea obtener parámetros.

```
// Obtiene los contextos de configuración
Configuration defaultConfiguration = GestorConfiguracion.getConfiguration("default");
Configuration pdfConfiguration = GestorConfiguracion.getConfiguration("pdf");
```

La referencia al objeto **Configuration** no debe ser almacenada por la aplicación. El componente GestorConfiguracion mantiene una cache con todos los contextos y realiza un uso eficiente de la memoria empleada por los mismos.

4.2.4 Obtención de parámetros de configuración

A partir de este momento ya puede leer los parámetros directamente del objeto Configuration.

```
// Accede a los parámetros de configuracion de la aplicación
String sistema = defaultConfiguration.getString("SISTEMA");
String fontSize = pdfConfiguration.getString("parameters.fonts.normal.size");
```

Para incorporar este código a su aplicación, añada las sentencias import correspondientes a los objetos Configuration y GestorConfiguración:

```
import org.apache.commons.configuration.Configuration;
import es.jcyl.configuracion.GestorConfiguracion;
```

5 GUÍA DE REFERENCIA

Como complemento al capítulo anterior, en este se van a presentar de forma detallada cada una de las características básicas asociadas a los elementos que se indicaron en el capítulo anterior.

5.1 Descripción del repositorio

Cómo se ha explicado de forma sencilla en el capítulo anterior, el funcionamiento del componente de configuración se basa en el fichero XML ConfigurationRespository.xml que a continuación se explicará en mayor detalle.

Como se puede observar fácilmente en el ejemplo de fichero de configuración presentado anteriormente, dentro del mismo se pueden distinguir básicamente dos elementos principales: “globals” y “contexts” los cuales se especificarán con detalle a continuación.

5.1.1 Globals

De forma general, puede definirse como el elemento contenedor de las propiedades globales del componente de configuración.

```
<!-- Configuraciones globales -->
<globals>

   <!-- Ruta base para todos los ficheros de configuración -->
   <initialPath value="/datos/aplicaciones/tb-solutions/configurador/" />

   <!-- Propiedades globales -->
   <properties>

       <property
           name="minimumCheckDelay"
           value="20000"
           comment="Tiempo de recarga de las propiedades de configuración (un valor de 0
equivale a no recargar). Expresado en milisegundos." />

       <property
           name="contextTimeOut"
           value="20000"
           comment="Tiempo que el sistema mantiene el contexto en memoria (un valor de 0
equivale a no recargar). Expresado en milisegundos." />

   </properties>

</globals>
```

Por un lado se encuentra el elemento “initialPath” y por el otro, anidadas en “properties” las diferentes propiedades que utiliza a nivel interno el componente:

- **InitialPath**: contiene la ruta absoluta a partir de la cual se localizan los ficheros relacionados en el repositorio. Este valor se define para facilitar la definición de las rutas de los ficheros de configuración, permitiendo que estas puedan ser expresadas de forma relativa.

- **Propiedad "minimumCheckDelay"**: Intervalo de tiempo que deja pasar el sistema antes de comprobar si ha cambiado un fichero de configuración. En caso de que haya cambiado se recargará de forma transparente al usuario. Este se expresa en milisegundos, interpretando que un valor de 0 equivale a indicar que no se lleva a cabo la recarga.

  Este intervalo debe definirse más o menos corto en función de la frecuencia con que se modifiquen los datos. A continuación se describen unas recomendaciones básicas:

Tipo de configuración	Recomendación
No cambia nunca	Establecer el valor "0" para indicar que no se debe compruebar la configuración.
Cambia con poca frecuencia	Establecer un intervalo no superior a 5 minutos entre recargas. Un valor muy elevado en este intervalo puede hacer que el instante de recarga sea dificil de predecir, ya que dependerá del instante en que se inició la aplicación.
Cambia con frecuencia	Establecer un intervalo de comprobación de 60 segundos. La comprobación cuando no hay cambios apenas tendrá coste, y a cambio se recargan los cambios con bastante frecuencia.
Tiempo real	El valor del intervalo puede establecerse todo lo pequeño que se quiera, para conseguir que los cambios se recarguen de forma casi instantánea. En la práctica valores inferiores a 5 segundos no tienen mucho sentido.

- **Propiedad "contextTimeOut"**: Indica el tiempo que el contexto es mantenido en memoria antes de ser desalojado por falta de uso. Al igual que el anterior, se expresa en milisegundos, y el valor 0 implica una no recarga y por tanto independencia de esta propiedad.

  El valor de este parámetro depende de las limitaciones de consume de memoria y CPU que vayan a imponerse a la aplicación. En general puede pensarse en 3 opciones:

Restricciones de rendimiento	Recomendación
No hay	Establecer el valor "0" para que no se desalojen los contextos. Se evitará el coste

	de carga y descarga de los contextos, a cambio de un mayor consumo de memoria.
Uso normal	Un tiempo de desalojo en condiciones de operación normal, en la que los parámetros de configuración hacen referencia únicamente a valores muy específicos, puede establecerse como proporcional al tiempo de recarga. De esta forma, si la configuración se comprueba cada 60 segundos parece razonable establecer el tiempo de desalojo en 60*10 = 600 segundos, por ejemplo.
Memoria limitada	En caso de que desee optimizarse el uso de la memoria, el valor recomendado es poner como timeout el doble del intervalo de recarga.

5.1.2 Contexts

El elemento “contexts” se define como un contenedor de los diferentes elementos de configuración que mantiene el componente, denominados de forma genérica contextos.

```
<!-- Lista de contextos de la aplicación -->
<contexts>

   <!-- Configuración de la aplicación -->
   <context id="default" name="Configuración de la aplicación">

       <!-- Fichero de configuración base -->
       <configurationFile
           id="application"
           name="Parámetros generales de la aplicación"
           location="/pruebas/app-config.properties"
           minimumCheckDelay="10000"
       />

       <!-- Fichero de despliegue de la aplicación -->
       <configurationFile
           id="webxml"
           name="Fichero de despliegue web.xml"
           location="/java/oc4j/j2ee/home/applications/demo/demo/WEB-INF/web.xml"
       />

   </context>

   <!-- Configuración del módulo de PDF (se desaloja a los 5 segundos) -->
   <context  id="pdf" name="Componente de generación de documentos PDF"
contextTimeOut="5000">

       <!-- Configuración del componente -->
       <configurationFile
           id="global"
           name="Parámetros generales del componente"
           location="./pdf.properties"
       />

   </context>
</contexts>
```

Dentro de él se definirán tantos elementos “context” como contextos sean gestionados, estableciendo como restricción que al menos tiene que existir uno, identificado como “default”.

Dentro de este contexto *por defecto* se incluirán aquellos ficheros generales de la aplicación, con el fin de garantizar la compatibilidad de la solución a la hora de integrarla en aplicaciones construidas anteriormente bajo el framework de desarrollo de la Junta de Castilla y León.

Cada uno de estos elementos “context” define los siguientes atributos:

- **id**: Define el identificador único del contexto dentro del repositorio. A través de él es identificado por el componente. A la hora de definir la identificación de un contexto, hay que considerar la restricción anterior al respecto de la necesidad de disponer siempre de un contexto “default”.
- **name**: Almacena el nombre del contexto. Se define con propósito informativo de cara al usuario.
- **minimumCheckDelay**: Intervalo de tiempo que deja pasar el sistema antes de comprobar si ha cambiado un fichero de configuración. En caso de que haya cambiado se recargará de forma transparente al usuario. Este valor tiene preferencia sobre el definido en las propiedades globales.

- **contextTimeout**: Indica el tiempo que el contexto es mantenido en memoria antes de ser desalojado por falta de uso. Este valor tiene preferencia sobre el definido en las propiedades globales.

Además de esta información, este elemento contiene la referencia a los diferentes ficheros que forman parte de él.

Estos son representados como elementos **configurationFile** y se definen mediante los siguientes atributos:

- **id**: Identificador único para el fichero dentro del contexto.
- **name**: Nombre identificativo del fichero, utilizado como valor informativo de cara al usuario.
- **location**: Ruta de localización del fichero, expresado en forma absoluta o relativa a la propiedad **initialPath**. Este componente soporta ficheros de configuración .properties y XML.
- **minimumCheckDelay**: Intervalo de tiempo que deja pasar el sistema antes de comprobar si ha cambiado un fichero de configuración. En caso de que haya cambiado se recargará de forma transparente al usuario. Este valor tiene preferencia sobre el definido en las propiedades globales y sobre el definido en el contexto.
- **comment**: se utiliza con el fin de definir en él otros comentarios que pudieran resultar relevantes para el fichero.

5.2 Acceso a las Propiedades del Repositorio

Una vez explicado detalladamente el contenido del fichero de configuración, se puede entender de una forma más clara como el componente va a manejar toda la información que necesita para gestionar el repositorio de contextos de la aplicación.

5.2.1 GestorConfiguración

Cómo se indicó en el capítulo anterior, el acceso al componente se lleva a cabo a través de una clase estática (GestorConfiguración) que actúa a modo de interfaz de comunicación entre la aplicación y el propio componente.

El uso de la clase se hace de forma sencilla, ya que sólo es necesario un paso previo de inicialización para poder acceder a las diferentes opciones del componente.

initialize

La llamada a este método define el proceso global de inicialización del componente. Se le pasa como parámetro la ruta de localización del fichero ConfigurationRepository.xml y a partir de ella el método desarrollará los siguientes pasos:

- Carga una única vez el fichero que define el repositorio a partir de la ruta indicada.
- Inicia una cache de contextos, que irá cargándolos a demanda de la aplicación y descargándolos cuando ésta no los use.

Ejemplo:

```
// Inicializa GestorConfiguracion a partir de la ruta donde se encuentra
GestorConfiguracion.initialize( ruta );
```

getConfiguration

Una vez inicializado el componente, los accesos a la información del mismo se llevarán a cabo a través de este método. Recibe como parámetro el identificador del elemento de contexto a recuperar y desencadena los siguientes pasos:

- Si el objeto Configuration se encuentra en cache lo devuelve.
- En caso contrario localiza los ficheros de configuración correspondientes al contexto, los lee desde la ubicación donde se encuentren, y devuelve un objeto Configuration con todos ellos.
- En cualquiera de los dos casos, se establece un tiempo de caducidad para ese contexto mediante el atributo **contextTimeOut** del repositorio o el específico para el propio contexto. Si transcurrido ese tiempo no se ha hecho uso del mismo será desalojado de memoria.

Ejemplo:

```
// Obtiene el contexto de configuración
Configuration configuration = GestorConfiguracion.getConfiguration("default");
```

5.2.2 El objeto org.apache.commons.configuration.Configuration

El objeto retornado por el componente GestorConfiguracion permite el acceso a las diferentes propiedades de configuración de la aplicación.

Este objeto pertenece a la librería “Commons Configuration” del proyecto Jakarta, y presenta una interfaz muy flexible para el acceso a los valores que almacena. Puede consultar más información sobre el mismo en el portal del proyecto:

http://jakarta.apache.org/commons/configuration/

Un ejemplo sencillo de uso es el descrito anteriormente para el acceso a una propiedad de tipo cadena:

```
// Accede a los parámetros de configuracion de la aplicación
String sistema = configuration.getString("SISTEMA");
```

También puede emplear este objeto para implementar soporte a propiedades distribuidas, de forma que uno de los contextos tenga prioridad sobre el otro:

```
// Obtiene los contextos de configuración
Configuration defaults = GestorConfiguracion.getConfiguration("default");
Configuration services = GestorConfiguracion.getConfiguration("services");

boolean enabled = true;

if (defaults.containsKey("SERVICE_DISABLED")) {
   enabled = defaults.getBoolean("SERVICE_DISABLED");
}
else if (services.containsKey("SERVICE_DISABLED")) {
   enabled = services.getBoolean("SERVICE_DISABLED");
}
```

Para aprovechar al máximo las funcionalidades de optimización de memoria que incorpora este componente, se recomienda que en ningún momento se almacene la referencia al contexto obtenido mediante el método getConfiguration(). El componente GestorConfiguracion ya incorpora un sistema interno de cache.

A continuación se incluyen ejemplos de diferentes alternativas de uso del componente Configuration, extraidas directamente de la web del proyecto.

5.2.3 Uso del componente con ficheros de propiedades

El objeto Configuration proporciona mecanismos para acceder a ficheros de propiedades que mejoran sustancialmente la clase `java.util.Properties` incluida en la distribución estándar de Java.

5.2.3.1 Includes

Si el fichero de propiedades contiene una propiedad de nombre "`include`" y el valor de la misma es un fichero en el disco, entonces el contenido de ese fichero será incluido como parte del objeto Configuration. Por ejemplo:

```
# usergui.properties
include = colors.properties
include = sizes.properties

# colors.properties
colors.background = #FFFFFF
```

5.2.3.2 Listas y arrays

Si necesita establecer una lista de valores en un fichero de propiedades, puede acceder a él posteriormente como si fuera un array en lugar de cómo un String.

Para ello, especifique los valores deseados como una lista separada por comas.

```
# chart colors
colors.pie = #FF0000, #00FF00, #0000FF
```

O repitiendo la propiedad tantas veces como desee, conservando el mismo nombre en varias líneas sucesivas:

```
# chart colors
colors.pie = #FF0000;
colors.pie = #00FF00;
colors.pie = #0000FF;
```

Posteriormente puede acceder a esos valores mediante el método getStringArray().

```
String[] colors = config.getStringArray("colors.pie");
```

5.2.3.3 Interpolación de variables

Si desea establecer una propiedad a modo de pseudo-variable con el fin de simplificar el proceso de modificar la configuración de una aplicación, puede hacerlo de la siguiente forma: En primer lugar declare la propiedad en el fichero:

```
application.name = GCON
application.version = 1.0
```

A partir de ese punto puede hacer referencia a su valor empleando la sintaxis ${nombre}.

```
application.title = ${application.name} ${application.version}
```

5.2.3.4 Caracteres especiales

Si desea indicar caracteres especiales en un fichero de propiedades (saltos de línea, tabuladores, etc.) puede hacerlo empleando la sintaxis propia de Java para códigos de escape.

Observe en el ejemplo que esta es la forma de declarar símbolos propios del fichero de configuración como la “coma”.

```
key = Esta \n cadena \t contiene \, caracteres \\ de control \u0020
```

5.2.4 Uso del componente con ficheros XML

Esta sección explica cómo usar ficheros XML jerárquicos y estructurados para especificar información de configuración de las aplicaciones.

5.2.4.1 Acceso a propiedades definidas en documentos XML

A continuación se representa un documento XML sencillo, que se va a utilizar como ejemplo para el acceso a las propiedades que almacena.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<gui-definition>
  <colors>
    <background>#808080</background>
    <text>#000000</text>
    <header>#008000</header>
    <link normal="#000080" visited="#800080"/>
    <default>${colors.header}</default>
  </colors>
  <rowsPerPage>15</rowsPerPage>
  <buttons>
    <name>OK,Cancel,Help</name>
  </buttons>
  <numberFormat pattern="###\,###.##"/>
</gui-definition>
```

Las propiedades almacenadas en este fichero pueden ser obtenidas desde el objeto Configuration utilizando técnicas muy variadas:

```
// Acceso a propiedades sencillas
String backColor = config.getString("colors.background");
String textColor = config.getString("colors.text");

// Acceso a propiedades usando atributos
String linkNormal = config.getString("colors.link[@normal]");

// Acceso a propiedades declaradas como pseudo-variables
String defColor = config.getString("colors.default");

// Lectura de valores numéricos
int rowsPerPage = config.getInt("rowsPerPage");

// Lectura de arrays y listas
List buttons = config.getList("buttons.name");
```

Cada uno de estos ejemplos introduce un aspecto destacado de la lectura de propiedades de configuración de un elemento xml:

- Los elementos anidados son accedidos usando una notación con puntos. Si un elemento **colors** tiene un elemento hijo **text**, la propiedad definida se llama **colors.text**.
- El elemento raíz del xml (en el ejemplo es **gui-definition**) se ignora en la regla anterior.
- Los atributos del XML se incluyen empleando una notación similar a la empleada por XPath.

- Puede usar interpolación de variables de la misma forma que con los ficheros de propiedades.
- Puede usar listas de propiedades de la misma forma que con los ficheros de propiedades.

5.2.4.2 Documentos XML complejos

Considere el siguiente escenario: Una aplicación opera con tablas de bases de datos y quiere cargar la definición del esquema de la base de datos de su configuración. El documento XML que proporciona esta información podría representarse de la forma siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<database>
  <tables>
    <table tableType="system">
      <name>users</name>
      <fields>
        <field>
          <name>uid</name>
          <type>long</type>
        </field>
        <field>
          <name>uname</name>
          <type>java.lang.String</type>
        </field>
        <field>
          <name>firstName</name>
          <type>java.lang.String</type>
        </field>
        <field>
          <name>lastName</name>
          <type>java.lang.String</type>
        </field>
      </fields>
    </table>
    <table tableType="application">
      <name>documents</name>
      <fields>
        <field>
          <name>docid</name>
          <type>long</type>
        </field>
        <field>
          <name>name</name>
          <type>java.lang.String</type>
        </field>
        <field>
          <name>creationDate</name>
          <type>java.util.Date</type>
        </field>
      </fields>
    </table>
  </tables>
</database>
```

El XML es auto-explicativo. Hay tantos elementos **table** como tablas hay en la base de datos, y cada uno de esos elementos tiene a su vez un elemento **field** para cada uno de los campos de la tabla.

No es posible acceder a los valores empleando una sintaxis similar a la anterior, pero el componente proporciona mecanismos alternativos.

En general cualquier valor que se repita en un fichero de configuración puede ser accedido en forma de lista. Esto se aplica para cualquier tipo de fichero de configuración.

Adicionalmente se soporta un mecanismo similar al de XPath para acceder a elementos por su posición.

La combinación de estas dos técnicas permite el acceso a los valores de un XML de múltiples formas.

Por ejemplo, para obtener los nombres de las tablas incluidas la sintaxis es:

```
// Acceso a los nombres de las tablas
Object prop1 = config.getProperty("tables.table.name");
if(prop1 instanceof Collection) {
   System.out.println("Tablas: " + ((Collection) prop1).size());
}
```

Por ejemplo, para obtener los nombres de los campos de la primera tabla la sintaxis es:

```
// Acceso a los nombres de los campos la primera tabla
Object prop2 =
config.getProperty("tables.table(0).fields.field.name");
if(prop2 instanceof Collection) {
   System.out.println("Campos de la primera tabla: " + ((Collection)
prop2).size());
}
```

También puede realizarse este acceso mediante el método **getList()** del objeto Configuration tal como se describe más arriba.

5.2.4.3 Puntuación en los nombres de los tags XML

Dado que el punto establece la forma de acceder a los elementos anidados en un XML, cuando este símbolo se encuentre presente dentro del nombre de un elemento deberá ser escapado correctamente.

Por ejemplo, para acceder al elemento test.value del siguiente documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<configuration>
  <test.value>42</test.value>
  <test.complex>
    <test.sub.element>many dots</test.sub.element>
  </test.complex>
</configuration>
```

La sintaxis es:

```
// Forma incorrecta
String complex = config.getString("test.value");

// Forma correcta
String complex = config.getString("test..value");
```

5.2.5 La clase es.jcyl.framework.JCYLConfiguracion

Esta clase define una interfaz de interacción con el componente que permite garantizar la compatibilidad de aplicaciones anteriores desarrolladas con el Framework de la Junta de Castilla y León.

Su comportamiento es similar al de versiones anteriores de la clase JCYLConfiguración, pero se encuentra implementado internamente mediante el componente GestorConfiguración. Los valores de configuración que son accedidos por esta clase sólo pueden pertenecer al contexto "**default**". Por esta razón la presencia de este contexto es obligatoria

Debe tener en cuenta que los métodos etiquetados como "deprecated" se aconseja ir abandonando su uso, aunque no hay intención de eliminarles.