|  |
| --- |
| **Commons** |
| Guía de instalación y uso |
|  |
|  |
|  |
| **Eduardo Becerra y Tomás Couso** |
| **09/01/2012** |
|  |

Contenido

[Introducción 4](#_Toc331533775)

[Instalación 4](#_Toc331533776)

[Devtools 5](#_Toc331533777)

[Configuración de una DB 5](#_Toc331533778)

[Carga de información de las tablas 6](#_Toc331533779)

[Generación de código 7](#_Toc331533780)

[Utilización del framework 8](#_Toc331533781)

[Estructura de paquetes 8](#_Toc331533782)

[BaseAction 9](#_Toc331533783)

[Propiedades 9](#_Toc331533784)

[Métodos 9](#_Toc331533785)

[BaseManagerAction 11](#_Toc331533786)

[Propiedades 12](#_Toc331533787)

[Métodos 12](#_Toc331533788)

[Opciones de menú 15](#_Toc331533789)

[Acciones 16](#_Toc331533790)

[Ejemplos 18](#_Toc331533791)

[Impresión de PDF 19](#_Toc331533792)

[Generación de Excel 20](#_Toc331533793)

[Carga de ficheros al servidor 22](#_Toc331533794)

[Gestión de sesiones por usuarios 23](#_Toc331533795)

[Mantenimientos incluidos en el Framework (Básico) 24](#_Toc331533796)

[Usuarios 24](#_Toc331533797)

[Perfiles 25](#_Toc331533798)

[Parámetros 25](#_Toc331533799)

[Cambiar clave 26](#_Toc331533800)

[Lookups 27](#_Toc331533801)

[Leyenda del log 27](#_Toc331533802)

[Logs de usuario 28](#_Toc331533803)

[Logs 29](#_Toc331533804)

[Sesiones activas 29](#_Toc331533805)

[Mantenimientos incluidos en el Framework (Extendido) 30](#_Toc331533806)

[Estructura de la DB 30](#_Toc331533807)

# Introducción

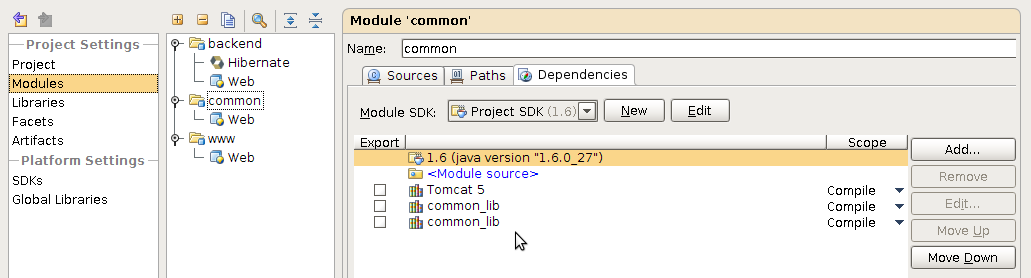
El framework “Commons” está formado por los siguientes elementos:

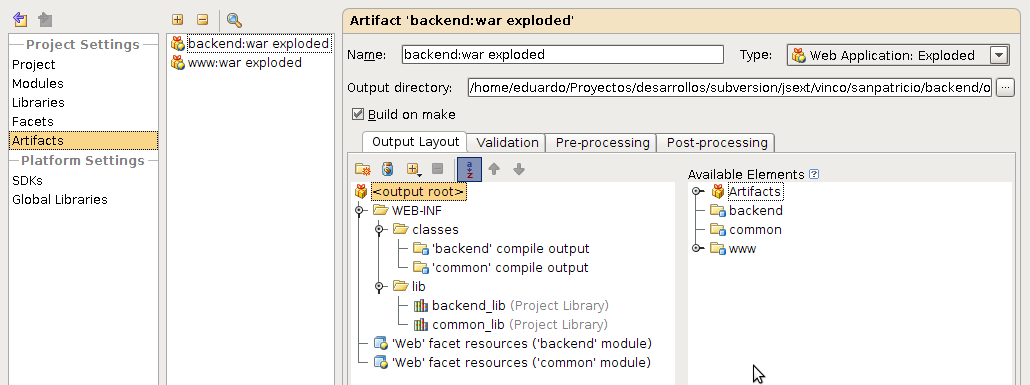
* *Base de datos*: Conjuntos de tablas que se utilizan para la configuración y trabajo del framework. Las tablas comienzan por APPBS\_ y APPEX\_, las tablas que comienzan por APPEX\_ son opcionales y no son necesaria crearlas en todos los entornos de trabajo.
* *Herramienta de desarrollo (****Devtools****)*: Es una aplicación Web que permite la conexión a las DB de desarrollo para la generación de código (Java, JSP, ExtJS, etc.) para el framework.
* *Biblioteca de Java (common)*: Clases de Java que forman en el framework commons.
* *Bibliotecas de Java*: Bibliotecas necesarias para el trabajo con el framework.
* *Bibliotecas de JavaScript*: Bibliotecas JavaScript que se utilizan en los JSP generados.

# Instalación

Para la instalación del framework se deben realizar los siguientes pasos:

* Creación de las tablas: En cada base de datos que se desee utilizar el framework se deben crear e inicializar las tablas necesarias. Utilizar los scripts correspondientes y aplicar todas las actualizaciones disponibles.
* Instalar el ***Devtools***: Consultar el capítulo dedicado al ***Devtools.***
* Creación del proyecto:
  + Utilizando un .jar del common: Incluir el jar en las libs del proyecto y utilizar una copia de los JSP
  + Utilizando los fuentes: Crear un modulo que utilice el common y configurar las dependencias correspondientes, así como el deployment.





# Devtools

El ***Devtools*** es una aplicación Web que permite la generación de código de forma automática para la gestión de los datos de las tablas de la DB configurada. Para la instalación del mismo se debe crearse la DB del devtools e instalar la aplicación en un servidor de aplicaciones.

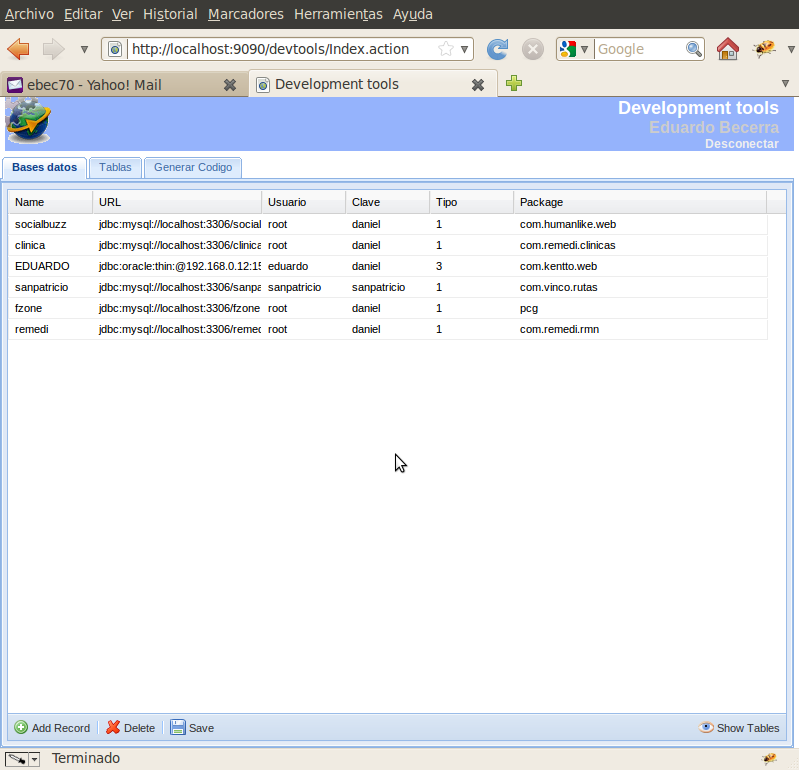
La aplicación se gestiona desde las siguientes pestañas:

* Bases de datos: Configuración de las bases de datos en las que se está desarrollando
* Tablas: Información de las tablas de la base de datos seleccionada
* Generar código: Se utiliza para la generación de código (acceso a datos, acciones, properties, JSP, ExtJS, etc).

## Configuración de una DB

Para configurar una conexión a una DB de desarrollo se necesitan los siguientes parámetros:

* Nombre: Nombre de la DB o esquema
* URL: Url (JDBC) de conexión a la DB
* Usuario: Usuario de la DB
* Clave: Clave de acceso del usuario a la DB
* Tipo: Tipo de la base de datos
  + 1. MySQL
  + 2. MSSQL Server
  + 3. Oracle
* Package: Nombre del paquete base (Java)

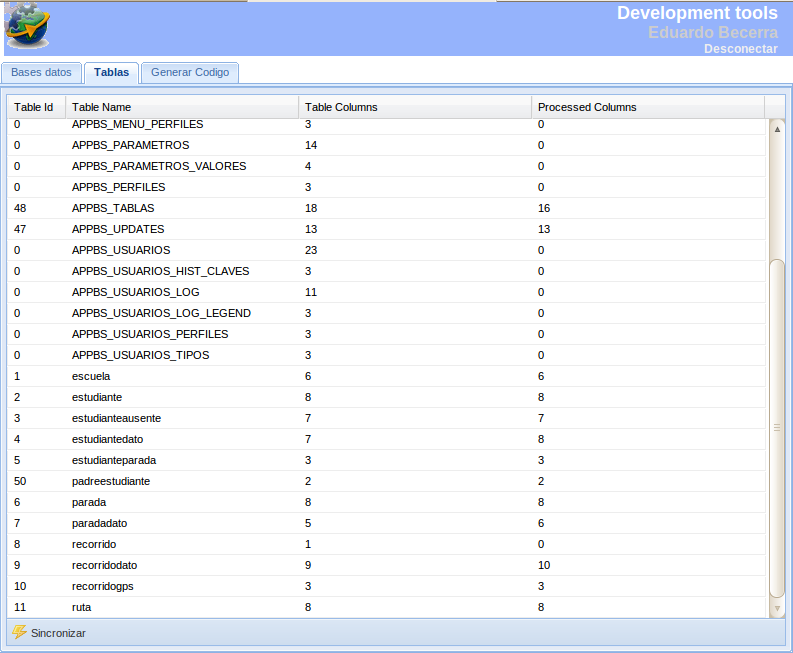


## Carga de información de las tablas

Una vez que se han definido las bases de datos que se van a utilizar para el desarrollo se puede cargar la información de las tablas (en las bases de datos configuradas es necesario que estén definidas las tablas del framework [APPBS\_\*]).

Seleccione una base de datos y presione el botón “Show Tables”, se mostrará la pestaña tablas con un listado de las tablas que tiene la base de datos (incluidas las del framework), mostrando el nombre, el numero de columnas de la tabla y el numero de columnas que se han cargado, en caso que no esté cargada la tabla (columna Table Id igual cero) o el numero de Table Columns y Processed Columns sea diferente se debe recargar la información de la tabla utilizando el botón “Sincronizar”.

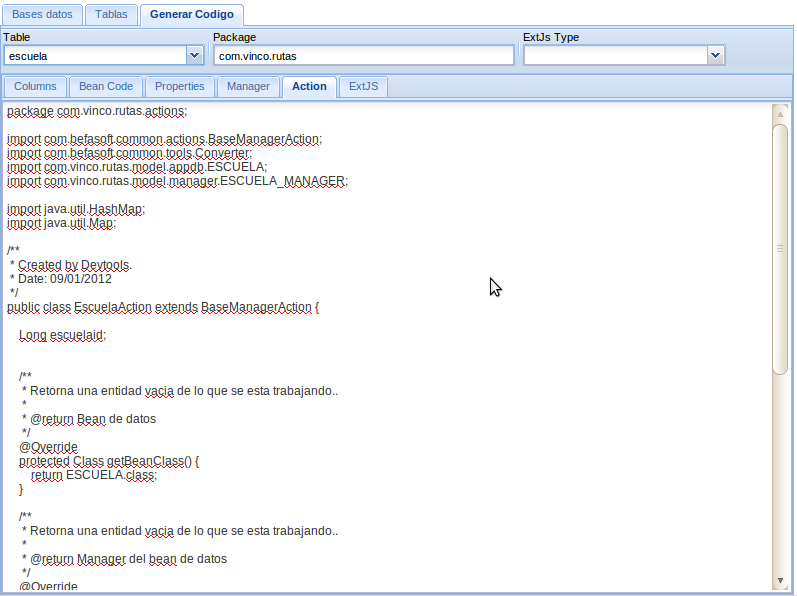
Para las tablas cargadas se pude utilizar la pestaña “Generar Código”.



## Generación de código

La pestaña de generación de código permite la generación de código de forma automática para las tablas cargadas. Se debe seleccionar una tabla y el código se genera de forma automática en las siguientes pestañas:

* Columns: Muestra la información cargada de la tabla seleccionada
* Bean Code: Genera el código para acceso a la tabla a través del Hibernate
* Properties: Genera las properties que se deben incluir en los ficheros de properties
* Manager: Genera el código para manipular los beans de la tabla
* Action: Genera el código del Acion (struts 2.0) que manipula las acciones sobre la tabla (listar, buscar, ordenar, insertar, actualizar, borrar)
* ExtJS: Genera el código ExtJS (JSP) para las operaciones de visualización y edición de los datos de la tabla. Se pueden generar los siguientes tipos de interfaz
  + Lista editable: Lista de elementos (Grid) editable
  + Lista con dialogo: Lista de elementos (Grid) que al seleccionar se abre un formulario de edición
  + Lista con formulario: Lista de elementos (Grid) con un panel adjunto para editar el elemento.



# Utilización del framework

El framework esta diseñado como un conjunto de clases que generalizan las principales tareas que se realizan en una aplicación Web tipo Backend, aunque también muchas funcionalidades se pueden aprovechar para aplicaciones de Frontend. De forma tal que se minimice el código necesario para desarrollar una aplicación.

A continuación explicaremos la estructura general de una aplicación y las principales clases que forman en core del framework.

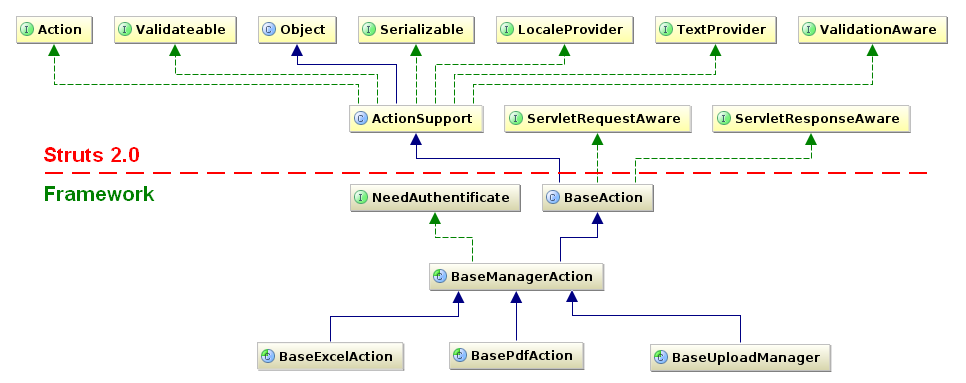
## Estructura de paquetes

Las clases de la aplicación se pueden definir dentro de cualquier espacio de nombres, pero dentro del mismo debe existir la siguiente estructura de paquetes (en negrita las obligatorias):

* **actions**: Se definen los actions de la aplicación Web (Struts 2.0)
* **model/appdb**: Beans de acceso a la DB (Hibernate)
* **model/manager**: Managers para operar sobre los datos
* **servlets**: Servlets que se utilicen en la aplicación, al menos se debe definir uno que se carga al iniciar la aplicación que debe heredar de ***com.befasoft.common.servlet.ConfLoader***
* **interceptors**: Interceptores de Struts 2.0

## BaseAction

La clase base de todas las acciones es BaseAction, después hay otras clases más especializadas, que veremos más adelante.



La clase BaseAction define el comportamiento básico de un action de Struts 2.0, además de incluir varias propiedades y métodos que facilitan el trabajo.

### Propiedades

|  |  |  |
| --- | --- | --- |
| Nombre | Tipo | Descripción |
| request | HttpServletRequest | Request de la petición HTTP |
| response | HttpServletResponse | Response de la petición HTTP |
| session | Map | Información de la session HTTP |
| action, show, op | String | Cadenas que se utilizan para intercambiar datos entre el action y el JSP |
| currentSession | Session | Sesión de trabajo de Hibernate |
| actionResponse | ActionResponse | Clase del framework con datos de respuesta |
| hasError | boolean | Bandera que indica si hubo error (app. Frontend) |
| user | LOGIN\_INFO | Información del usuario que esta logon en la aplicación |
| log | Log | Logger de la aplicación |

### Métodos

/\*\*

\* Escribe el el log del usuario

\* @param id\_tipo Id. del tipo (Lookup de la tabla APPBS\_USUARIOS\_LOG\_LEGEND

\* @param nivel Nivel de log

\* @param text1 Texto 1

\* @param text2 Texto 2

\* @param text3 Texto 3

\* @param text4 Texto 4

\* @param text5 Texto 5

\*/

public void ***writeUserLog***(Long id\_tipo, Long nivel, String text1, String text2, String text3, String text4, String text5);

Esta función permite escribir en las tablas de log del usuario, si la aplicación utiliza este método se puede tener un seguimiento de las acciones que realiza cada usuario en la aplicación.

/\*\*

\* Metodo que se llama por defecto al ejecutar la accion

\*

\* @return Retorna el valor que se utiliza para enlazar con el result de STRUTS

\* @throws Exception

\*/

public String ***execute***() throws Exception;

Este método siempre debe ser llamado por las acciones que hereden de esta clase, al terminar este método se pueden utilizar las propiedades: sesión, user y currentSession.

/\*\*

\* Retorna la información de usario para almacenar en la sesion

\* @param user Informacion del usario (tabla APPBS\_USUARIOS)

\* @return Informacion del usuario

\*/

public LOGIN\_INFO ***getLoginInfo***(APPBS\_USUARIOS user);

Retorna la información del usuario que se almacena en la sesión, lo habitual es definir una clase de información de usuario que herede de LOGIN\_INFO y retornar una instancia de la misma, esta instancia será almacena en la sesión y retornada en la propiedad ***user*** en cada llamada a un action.

/\*\*

\* Valida si el usario tiene acceso a la aplicacion

\* @param login Login del usuario

\* @param password Clave de acceso

\* @param remember Indica si se recuerda el usuario a través de Cookies

\* @return Informacion del usuario

\* @throws Exception Error en la DB

\*/

protected LOGIN\_INFO ***loginUser***(String login, String password, boolean remember) throws Exception;

protected LOGIN\_INFO ***loginUser***(String login, String password) throws Exception;

protected LOGIN\_INFO ***loginUser***(APPBS\_USUARIOS user, boolean remember) throws Exception;

protected LOGIN\_INFO ***loginUser***(APPBS\_USUARIOS user) throws Exception;

Verifica si el usuario tiene acceso a la aplicación. Retorna la información del usuario, el objeto ***actionResponse*** contiene los errores en caso que no se pueda validar el usuario. El parámetro ***remember*** indica si se desea que envíen cookies al cliente Web para hacer la validación automática en las próximas validaciones del usuario.

/\*\*

\* Valida si hay cookie de usuario almacenada

\* @return Verdadero si valido al usuario

\* @throws Exception Error en la DB

\*/

protected LOGIN\_INFO ***checkLoginCookie***() throws Exception;

Valida un usuario utilizando las cookies almacenadas en el cliente, si las cookies son correcta realiza la validación del usuario (***loginUser***).

/\*\*

\* Remueve la cookie de usuario almacenada

\* @throws Exception Error en la DB

\*/

protected void ***removeLoginCookie***() throws Exception;

Elimina la información de las cookies del usuario.

/\*\*

\* Recupera la clave de acceso

\* @param template Nombre del template html

\* @param email Direccion de email

\* @throws Exception Errores

\* @return Indica si se envio la solicitud correctamente

\*/

protected boolean ***recoverPassword***(String template, String email) throws Exception;

Inicia el proceso de recuperación de la clave del usuario. Envía un email al usuario con las indicaciones de para resetear la clave, para enviar el email se utiliza un *StringTemplate* que tiene que tener dos campos: link (se envía el enlace para recuperar la contraseña) y nombre (personalización del email con el nombre del usuario). En ***actionResponse*** se incluye un mensaje de con el éxito o fracaso de la acción.

/\*\*

\* Inicia el reseteo de la contraseña

\* @param email Direccion de email

\* @param hash Hash de seguridad

\* @param fecha Fecha de solicitud

\* @return Id. del usuario

\*/

protected APPBS\_USUARIOS ***resetPassword***(String email, String hash, String fecha);

Inicia el proceso de cambio de la contraseña del usuario, verificando que la solicitud fue enviada desde la aplicación por el propio usuario (se utilizan funciones estándar criptográficas seguras). En ***actionResponse*** se incluyen los mensajes de error correspondiente.

/\*\*

\* Cambia la clave de acceso de un cliente

\* @param id\_usuario Id. del usuario

\* @param password Nueva clave

\*/

public void ***changePassword***(Long id\_usuario, String password);

Cambia la clave de acceso de un usuario. Para llevar un registro de los cambios de claves realizados, así como validando que se cumplan las políticas de cambio de contraseñas. Se debe utilizar la clase ***com.befasoft.common.actions.PasswordAction***.

/\*\*

\* Selecciona el idioma por defecto del navegador (se utiliza el siguiente orden de seleccion)

\*

\* 1. Primer idioma disponible de la lista de idiomas del navegador

\* 2. Idioma por defecto

\* 3. Primer idioma disponible

\*/

public void ***loadNavLanguage***();

Carga el idioma según las preferencias del navegador del cliente. En la sesión se almacena la siguiente información: idioma -> Datos con el idioma (APPEX\_IDIOMAS), id\_idioma -> Id. del idioma

## BaseManagerAction

La clase ***BaseManagerAction*** hereda directamente de ***BaseAction*** y adiciona las funcionalidades de gestión de datos: listar, filtrar, insertar, modificar y borrar. Además de otras funcionalidades que amplían las funcionalidades de un action. Las action que se generan a través de del ***Devtools*** extienden de esta clase. Además esta clase implementa la interfaz ***NeedAuthentificate*** que indica si para ejecutar la acción es necesario estar autentificado como un usuario registrado.

/\*\*

\* Retorna si es necesario que el usuario este autentificado para realizar esta accion.

\*

\* @return <code>true</code> si es necesario estar autentificado

\*/

public boolean ***needAuthetificated***();

### Propiedades

|  |  |  |
| --- | --- | --- |
| Nombre | Tipo | Descripción |
| bodyResult | BodyResult | Similar a ***actionResponse*** pero mucho más potente |
| elementsToDelete | String | Cadena en formato JSON con la lista de elementos a borrar |
| dataToSave | String | Cadena en formato JSON con la lista de elementos a modificar/insertar |
| limit | int | Cantidad de elementos a retornar en un listado (Paginado) |
| start | int | Primer elemento a retornar de la lista (Paginado) |
| sort | String | Campo por el que se ordena la lista de elementos |
| dir | String | Dirección de ordenación (ASC, DESC) |
| element | EntityBean | Datos de un elemento |

### Métodos

/\*\*

\* Borra los elementos seleccionados

\*

\* @param bean Bean a borrar

\* @param dataList Datos a borrar

\*/

protected void ***deleteBeanElements***(EntityBean bean, List<HashMap> dataList);

Borra los elementos que se pasan como argumento. Para cada elemento se llama a la función ***canDelete*** si retorna un error, no se borra el elemento. Al final el borrado de los elementos se llama a la función ***wasDeleted*** para cada elemento borrado.

/\*\*

\* Actualiza los elementos

\*

\* @param bean Bean a actualizar

\* @param dataList Lista de objectos a actualizar

\*/

protected void ***updateBeanElements***(EntityBean bean, List<HashMap> dataList);

Actualiza/inserta los elementos que se pasan como argumentos. Para cada elemento se llaman a los métodos ***canInsert*** o ***canModify*** según corresponda. Para cada elemento modificado se llama a la función ***wasModified***.

/\*\*

\* Metodo usado para validar los campos antes de guardarlos en BD y poder retornar errores asociados a cada campo.

\*

\* @param bean Bean a actualizar

\* @param dataList Lista de objectos a actualizar

\* @return <code>true</code> Si es valido el bean

\*/

protected boolean ***validateSave***(EntityBean bean, List<HashMap> dataList);

/\*\*

\* Valida cada campo del bean en este metodo solo se valida los valores de cada property

\*

\* @param bean Bean a actualizar

\* @param dataRow Hash con los datos

\* @return <code>true</code> Si es valido el bean

\*/

protected boolean ***validateBeanSave***(EntityBean bean, HashMap dataRow);

Antes de guardar los cambios se validan que los datos no tengan errores, aunque normalmente las validaciones se hacen en la parte del cliente.

/\*\*

\* Metodo que llama al Execute del padre

\*

\* @return Retorna el valor que se utiliza para enlazar con el result de STRUTS

\* @throws Exception Error

\*/

public String ***executeInit***() throws Exception;

Inicializa el Log y llama al ***execute*** del ***BaseAction***. Se puede utilizar cuando se desea anular el comportamiento por defecto del ***execute***.

/\*\*

\* Metodo que se llama por defecto al ejecutar la accion

\*

\* @return Retorna el valor que se utiliza para enlazar con el result de STRUTS

\* @throws Exception Error

\*/

public String ***execute***() throws Exception;

Ejecución del action, según los parámetros que se le pase, listara, filtrara, actualizará, insertará o borrara datos. Si el parámetro action tiene el nombre de un método lo llamara directamente.

/\*\*

\* Prepara los resultado

\* @param allElements Lista de elementos

\* @param totalCount Cantidad total de elementos (Paginado)

\* @param start Posicion del primer elemento (Paginado)

\* @param limit Cantidad de elementos (Paginado)

\*/

protected void ***setReturn***(List allElements, int totalCount, int start, int limit);

protected void ***setReturn***(List allElements);

protected void ***setReturn***(List allElements, int totalCount);

protected void ***setReturn***(Object element);

Este método utilizando los parámetros prepara la respuesta (***bodyResult***).

/\*\*

\* Retorna el mapa de los parametros del filtro

\*

\* @return Filtros

\*/

protected Map ***getFilter***();

Este método retorna un filtro para aplicar al listado de elementos que retorna el action. En el próximo capítulo veremos un ejemplo práctico.

/\*\*

\* Retorna la clase asociada al bean..

\*

\* @return Bean de datos

\*/

protected abstract Class ***getBeanClass***();

Retorna la clase que gestiona la table (Hibernate).

/\*\*

\* Retorna la clase asociada al manager del bean..

\*

\* @return Manager del bean de datos

\*/

protected abstract Class ***getManagerClass***();

Retorna el Manager que gestiona la tabla.

/\*\*

\* Ejecuta guardar una entidad.

\*

\* @return Action result

\* @throws Exception Error

\*/

protected String ***executeUpdate***() throws Exception;

Realiza la actualización de los datos (modificar/insertar)

/\*\*

\* Ejecuta guardar una entidad.

\*

\* @return Action result

\* @throws Exception Error

\*/

protected String ***executeDelete***() throws Exception;

Realiza el borrado de los datos.

/\*\*

\* Indica si se puede borrar este registro

\*

\* @param values Map con los valores recibidos

\* @param bean Bean que se va a borrar

\* @return <code>true</code> Si se puede borrar este registro

\* @throws Exception Error

\*/

protected String ***canDelete***(Map values, EntityBean bean) throws Exception;

Es llamado antes de borrar un registro. En este método se pueden realizar tareas previas al borrado del registro o verificar si puede borrarse o no. Retorna un error, en caso que se pueda continuar el proceso de borrado deberá retornar **null**.

/\*\*

\* Indica si se puede modificar este registro

\*

\* @param values Map con los valores recibidos

\* @param bean Bean que se va a modificar

\* @return <code>true</code> Si se puede modificar este registro

\* @throws Exception Error

\*/

protected String ***canModify***(Map values, EntityBean bean) throws Exception;

Similar a ***canDelete***, pero para los registros que se modifican.

/\*\*

\* Indica si se puede insertar este registro

\*

\* @param values Map con los valores recibidos

\* @param bean Bean que se va a insertar

\* @return <code>true</code> Si se puede insertar este registro

\* @throws Exception Error

\*/

protected String ***canInsert***(Map values, EntityBean bean) throws Exception;

Similar a ***canDelete***, pero para los registros que se insertan.

/\*\*

\* Se llama cuando el registro ha sido modificado

\*

\* @param bean Bean modificado

\* @throws Exception Error

\*/

protected void ***wasModified***(EntityBean bean) throws Exception;

Este método se llama para los registro que se han modificado/insertado, en el se pueden realizar tareas posteriores a la modificación del registro.

/\*\*

\* Se llama cuando el registro ha sido borrado

\*

\* @param bean Bean borrado

\* @throws Exception Error

\*/

protected void ***wasDeleted***(EntityBean bean) throws Exception;

Similar al anterior para los registros borrados.

/\*\*

\* Ejecuta guardar una entidad.

\*

\* @return Action result

\* @throws Exception Error

\*/

public String ***executeSave***() throws Exception;

Aplica las modificaciones.

/\*\*

\* Retorna el listado de elementos siguiente el filtro y el paginado

\*

\* @return Lista de elementos

\* @throws Exception Error en la llamada a los bean

\*/

protected String ***executeList***() throws Exception;

Ejecuta el filtrado y listado de registros.

## Opciones de menú

El menú de la aplicación se gestiona a través de las tablas: APPBS\_MENU y APPBS\_MENU\_PERFILES (esta tabla puede controlarse desde el mantenimiento de perfiles de usuario), como las opciones de menú están estrechamente vinculadas a la programación de las acciones propias del menú, el menú se gestiona directamente actualizando las tablas correspondiente en la DB. La tabla APPBS\_MENU tiene muchos campos, pero en este ejemplo nos concentraremos en los imprescindibles, los otros campos están disponibles para personalizaciones del framework.

Por ejemplo si quisiéramos inserta un menú Clientes con las opciones Mantenimiento e Informes, para el perfil de usuario 1.

Se inserta la opción de menú principal, ID\_PADRE = 'MAIN\_MENU'

INSERT INTO APPBS\_MENU (ID\_APLICACION, ID\_MENU, NOMBRE, ORDEN, ID\_PADRE, TIPO\_ENLACE) VALUES ('DEMO', 'CLIENTES', 'Clientes', '100', 'MAIN\_MENU', 'N');

Se insertan las opciones del submenú, ID\_PADRE = 'CLIENTES'

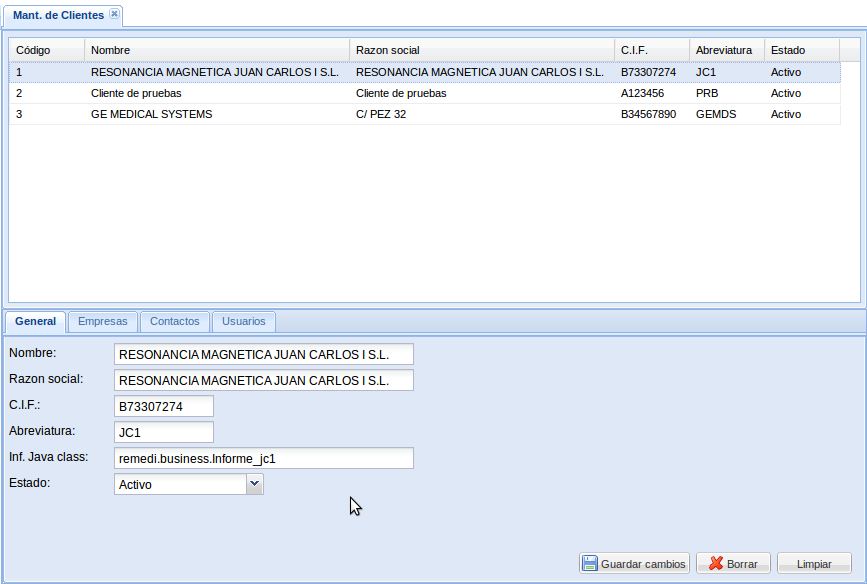
INSERT INTO APPBS\_MENU (ID\_APLICACION, ID\_MENU, NOMBRE, ORDEN, ID\_PADRE, TIPO\_ENLACE) VALUES ('DEMO', 'CLIENTES\_MANT', 'Mantenimiento', '100', 'CLIENTES', 'N'), ('DEMO', 'CLIENTES\_INF', 'Informes', '200', 'CLIENTES', 'N');

Opcionalmente se puede adicionar los perfiles para los que están disponibles estas opciones de menú.

INSERT INTO APPBS\_MENU\_PERFILES (ID\_APLICACION, ID\_MENU, ID\_PERFIL) VALUES ('DEMO', 'CLIENTES', 1), ('DEMO', 'CLIENTES\_MANT', 1), ('DEMO', 'CLIENTES\_INF',1);

## Acciones

En esta sección explicaremos como definir acciones, utilizando ejemplos reales. En este primer ejemplo queremos hacer el mantenimiento de la tabla CLIENTES, que tiene varios campos que definen un cliente. El *action* que permite el mantenimiento de la tabla será el que genera directamente el Devtools, quitando los filtros ya que siempre se muestran todos los clientes.



package com.remedi.rmn.actions;

import com.befasoft.common.actions.BaseManagerAction;

import com.remedi.rmn.model.appdb.CLIENTES;

import com.remedi.rmn.model.manager.CLIENTES\_MANAGER;

/\*\*

\* Created by Devtools.

\* Date: 04/11/2011

\*/

public class ClientesAction extends BaseManagerAction {

/\*\*

\* Retorna una entidad vacia de lo que se esta trabajando.

\*

\* @return Bean de datos

\*/

@Override

protected Class ***getBeanClass***() {

return CLIENTES.class;

}

/\*\*

\* Retorna una entidad vacia de lo que se esta trabajando.

\*

\* @return Manager del bean de datos

\*/

@Override

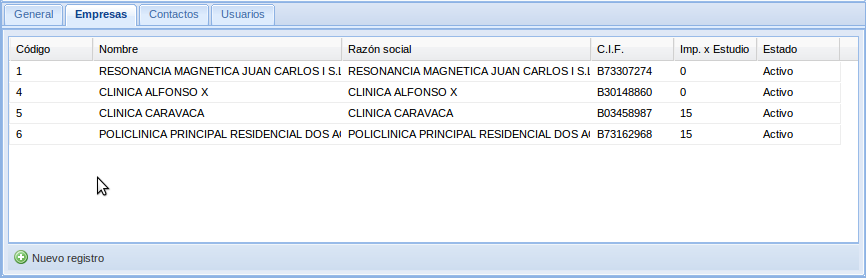
protected Class ***getManagerClass***() {

return CLIENTES\_MANAGER.class;

}

}

Este es el *action* más simple que podemos definir, pero es completamente funcional y realiza las tareas de listar, modificar, insertar y borrar.



En este ejemplo un cliente tiene asociadas varias EMPRESAS, a continuación veremos el *action* que gestiona las empresas.

package com.remedi.rmn.actions;

import com.befasoft.common.actions.BaseManagerAction;

import com.befasoft.common.tools.Converter;

import com.remedi.rmn.model.appdb.EMPRESAS;

import com.remedi.rmn.model.manager.EMPRESAS\_MANAGER;

import java.util.HashMap;

import java.util.Map;

/\*\*

\* Created by Devtools.

\* Date: 04/11/2011

\*/

public class EmpresasAction extends BaseManagerAction {

Long id\_cliente;

/\*\*

\* Retorna una entidad vacia de lo que se esta trabajando..

\*

\* @return Bean de datos

\*/

@Override

protected Class ***getBeanClass***() {

return EMPRESAS.class;

}

/\*\*

\* Retorna una entidad vacia de lo que se esta trabajando..

\*

\* @return Manager del bean de datos

\*/

@Override

protected Class ***getManagerClass***() {

return EMPRESAS\_MANAGER.class;

}

/\*\*

\* Retorna el mapa de los parametros del filtro

\*

\* @return Filtros

\*/

@Override

protected Map ***getFilter***() {

Map filter = new HashMap();

if (!Converter.isEmpty(id\_cliente))

filter.put("id\_cliente", id\_cliente);

return filter;

}

/\*

\* Metodos Get/Set

\*/

public Long ***getId\_cliente***() {

return id\_cliente;

}

public void ***setId\_cliente***(Long id\_cliente) {

this.id\_cliente = id\_cliente;

}

}

Este *action* es muy similar al de clientes, excepto que incorpora un parámetro ***id\_cliente*** por el que se filtra, el listado de empresas.

### Ejemplos

A continuación expondremos ejemplos a través de diferentes formas de realizar tareas hipotéticas.

public class EjemploAction extends BaseManagerAction {

protected String ***executeList***() throws Exception {

super.executeList();

for (int i = 0; i < bodyResult.getElements().size(); i++) {

// Opera sobre cada uno de los elementos que se listan

}

return Action.SUCCESS;

}

Después de ejecutar de listar los elementos realizar operaciones sobre cada uno de ellos, por ejemplo se listan los artículos y se comprueba si existen las fotos correspondientes en el sistema de ficheros.

protected String ***canDelete***(Map values, EntityBean bean) throws Exception { CLIENTES cliente = (CLIENTES) bean; String error = null; if (Constants.FIRST\_ID\_CLIENTES > cliente.getId\_cliente()) error = getText("clientes.err.delete.old"); else if ("N".equals(cliente.getExportable())) error = getText("clientes.err.export"); return error;}Verifica si el registro se puede borrar, en caso que se pueda borrar retorna **null** sino el mensaje de error con el motivo por el que no se puede borrar el registro.

private String ***listNovedades***() { List articulos = ARTICULOS\_MANAGER.findArticulos(start, limit, id\_idioma); setReturn(articulos); writeUserLog(Constants.LOG\_CATALOG\_NOVEDADES, APPBS\_USUARIOS\_LOG.LEVEL\_INFO); return Action.SUCCESS;

}

Lista los productos que son novedades, para que este método sea llamado el *action* tiene que invocarse desde el JSP como */Ejemplo.action?action= listNovedades*. Además se graba un registro en el log de usuario registrando la acción realizada.

}

## Impresión de PDF

La impresión de ficheros PDF se realiza utilizando la biblioteca *iText*, consultar la documentación correspondiente. La clase ***BasePDFAction*** encapsula la impresión de los PDF dentro de una aplicación Web y ofrece algunos métodos de soporte.

Para la impresión de ficheros PDF el Action debe heredar de la clase ***BasePDFAction***, redefiniendo los siguientes métodos:

/\*\*

\* Genera el PDF

\* @throws Exception Error

\* @return Verdadero si se genero correctamente el documento

\*/

abstract protected boolean ***writePdf***() throws Exception;

Dentro de este método se debe generar el documento PDF, por ejemplo:

protected boolean ***writePdf***() throws Exception { file = "Preview.pdf"; // Define el nombre del fichero

// Crea una table con dos columnas

PdfPTable table = new PdfPTable(new float[] {0.15f, 0.85f}); // En la primera celta alineada a la derecha y en negrita

table.addCell(

getCell(fontBold, Rectangle.ALIGN\_RIGHT, "Hello World!"));

// Segunda celda

table.addCell(

getCell(fontItem, Rectangle.ALIGN\_RIGHT, "Demo Text"));

document.add(table);

return true;

}

Se tienen disponibles los siguientes campos:

protected Font fontT1;

protected Font fontT2;

protected Font fontBold;

protected Font fontItem;

protected Font fontLite;

protected BaseColor bg;

protected Document document;

protected PdfWriter writer = null;

protected Rectangle pgSize;

protected String file;

protected boolean toFile = false;

protected boolean toRequest = true;

Los campos ***font\**** predefinen distintos tipos de font, si se desean cambiar los mismos se tienen los correspondientes métodos ***get***. Al igual que ***bg*** que define un color.

El campo ***document*** y ***writer*** dan acceso a las componentes correspondientes de *iText*.

Para dibujar rectángulos y líneas se tienen los métodos: *drawRoundRect*, *drawRect* y *drawLine*.

Para el trabajo con tablas y celdas se tienen varios métodos de soporte: *buildTable* , *getCell*, *getCellPadding* y *getExpandCell*

## Generación de Excel

La generación de ficheros Excel se realizando la biblioteca POI de Apache, consultar la documentación correspondiente. La clase ***BaseExcelAction*** encapsula la generación de ficheros Excel dentro de una aplicación Web y ofrece algunos métodos de soporte.

Los Actions que qnecesiten generar ficheros Excel deberán heredar de la clase ***BaseExcelAction*** e implementar el método:

/\*\*

\* Genera el Excel

\* @throws Exception Error

\* @return Verdadero si se genero correctamente el documento

\*/

abstract protected boolean ***writeExcel***() throws Exception;

Se tienen disponibles los siguientes campos:

protected HSSFWorkbook wb;

protected HSSFSheet sheet;

protected HSSFCellStyle csCaption;

protected HSSFCellStyle csColTitle;

protected HSSFCellStyle csBold;

protected HSSFCellStyle csNormal;

protected HSSFCellStyle csNormalRed;

protected HSSFCellStyle csNormalDate;

protected String sheetName = "Hoja1";

protected String file;

protected boolean toFile = false;

protected boolean toRequest = true;

Los campos ***cs\**** definen diferentes estilos para aplicar a las celdas, ***wb*** y ***sheet*** dan acceso al libro y la hoja respectivamente.

El método ***exportItems*** exporta un listado de beans a Excel. La clase de los beans a exportar debe implementar los siguientes métodos:

* *getExportFields*: Retorna una lista de los campos a exportar (columnas)
* *getFormulas*: Retorna un listado con las formulas (Opcional)
* *getColWidth*: Retorna un listado con los anchos de las columnas (Opcional)

Ejemplo:

@Entity

@Table(name = "ESTUDIANDES")

public class ESTUDIANDES extends EntityBean implements Cloneable {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(name = "ID\_ESTUDIANTE")

private Long id\_estudiante;

@Column(name = "NOMBRE")

private String nombre;

@Column(name = "APELLIDOS")

private String apellidos;

@Column(name = "SEXO")

private String sexo;

@Column(name = "EDAD")

private Long edad;

/\*\*

\* Nombre de los campos que se van a utilizar en una exportación

\*

\* @return Lista de campos

\*/

@JSON(serialize = false)

public String[] ***getExportFields***() {

return new String[] {

"nombre", "apellidos", "sexo", "edad"

};

}

}

public class EstudiantesAction extends BaseExcelAction {

/\*\*

\* Genera el Excel

\* @throws Exception Error

\* @return Verdadero si se genero correctamente el documento

\*/

protected boolean ***writeExcel***() throws Exception {

executeList();

List estList = bodyResult.getElements();

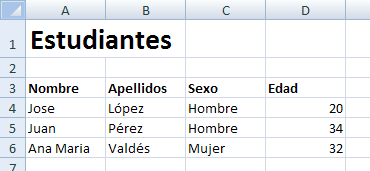
exportItems(getText("estudiantes.caption"), " estudiantes.", estList);

}

}

Para llamar a esta acción se debe abrir una ventana nueva en el navegador, especificando como ***action*** ***export***:

window.open( '/Estudiantes.action?***action=export'***);



## Carga de ficheros al servidor

Cuando se necesite cargar ficheros al servidor, el action que realiza la tarea debe heredar de la clase ***BaseUploadManager***. Se pueden redefinir los siguientes métodos para la personalización de la subida del fichero.

* *getDestFileName*: Camino completo del fichero donde se guardará el fichero que se está subiendo. Este es la única función que es obligatorio redefinir.
* *processFile*: Se llama esta función cuando se ha terminado de subir el fichero.
* *getProcessError*: En caso de error en el procesamiento del fichero se llama esta función para obtener el mensaje de error.
* *getSizeError*: Obtiene el mensaje de error cuando el fichero supera el límite establecido
* *isValidContentType*: Verifica que el contenido del fichero sea correcto
* *getContentTypeError*: Obtiene el mensaje de error cuando el tipo de fichero no es correcto.
* *getResultExtra*: Cadena con parámetros extras a pasar en los JSP

En los JSP que necesitan cargar el fichero, se debe incluir el fichero ***appbs\_extjs/loadfile.jsp*** que contiene las funciones y diálogos necesarios para cargar el fichero. Se debe llamar a la función ***appbs\_Loadfile*** con los siguientes argumentos:

* *urlAction*: Url del action que procesa el fichero
* *callback*: Función de javascript que se llama al terminar la carga
* *directory*: Directorio que se pasa al servidor
* *contentType*: Lista de tipos de contenido del fichero
* *maxSize*: Tamaño máximo del fichero

Ejemplo:

appbs\_Loadfile(

“Cargar.action?action=upload”,

function (params){

if (params.success) {

// Carga correcta

}

},

“artículos/fotos”,

“images/jpg;images/png;images/gif”,

2000000

);

## Gestión de sesiones por usuarios

En caso que se quiera llevar un control de las sesiones que inician los usuarios (el usuario solo puede entrar desde un solo navegador) es necesario cargar el **SessionManager** en al iniciar la aplicación para ello es necesario incluir la siguiente sección en el fichero web.xml.

<listener>

<listener-class>com.befasoft.common.servlet.SessionManager</listener-class>

</listener>

Y en los Actions que se procese el login de la aplicación redefinir las funciones siguientes:

/\*\*

\* Indica si se valida la multisesion

\* @return Verdarero si el usuario solo puede entrar una vez

\*/

public boolean ***checkUniqueSession***() {

return true;

}

/\*\*

\* Indica el modo de tratar las sesiones unicas

\* @return

\* true - La primera sesion es la que se mantiene, se rechaza el login

\* false - Se acepta el login y se invalidan el resto de sessiones activas

\*/

public boolean ***uniqueSessionFirst***() {

return true;

}

/\*\*

\* Indica si se permiten sesiones Web y Mobile al mismo tiempo

\* @return

\* true - Permite una sesion Web y otra Mobile

\* false - Solo permite una sesion

\*/

public boolean ***allowMobileAndWebSession***() {

return true;

}

Retornar true en la función ***checkUniqueSession***() para activar el control se sesiones y la función ***uniqueSessionFirst***() determina el modo en que se gestionan las sesiones simultaneas.

Además la función ***allowMobileAndWebSession***() indica si se permite una sesión simultanea desde la Web y otra desde un dispositivo móvil, en caso que se desee este comportamiento antes de llamar a las funciones de login (***validateUser***) se debe inicializar la variable ***mobile*** a ***true*** en la sesión que se inicia desde el dispositivo móvil.

# Mantenimientos incluidos en el Framework (Básico)

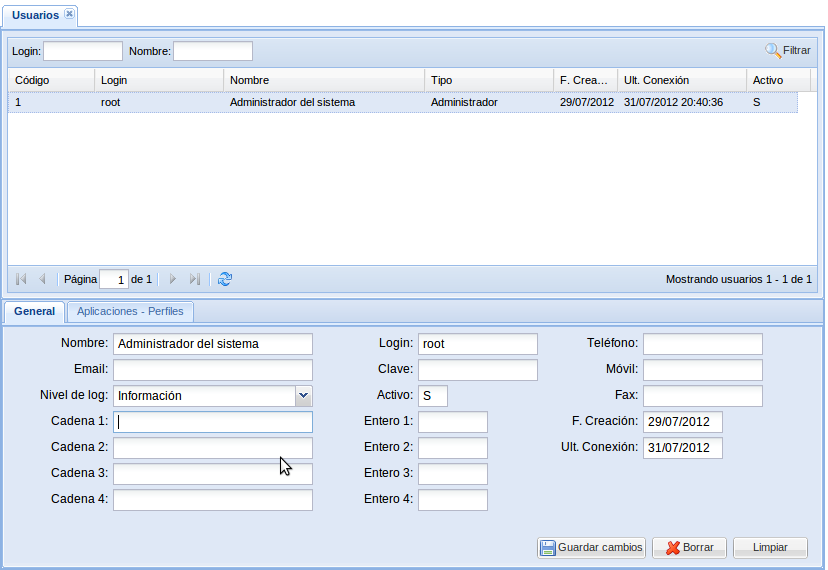
Existen implementados varios mantenimientos para gestionar las principales tablas del framework, a continuación detallaremos cada una de ellas.

Se incluyen los siguientes mantenimientos:

* Usuarios
* Perfiles
* Parámetros
* Cambiar clave
* Lookups
* Leyenda del log
* Logs
* Sesiones activas

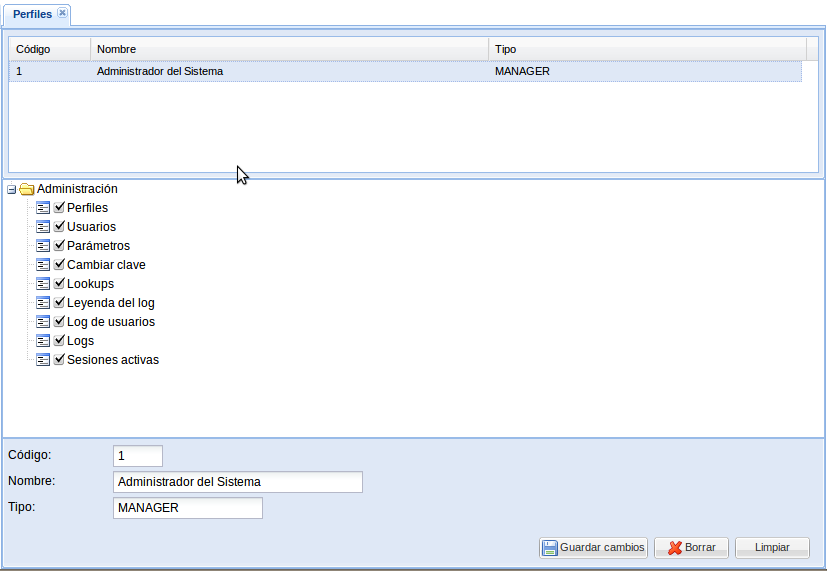
## Usuarios

Gestiona los usuarios que tienen acceso al sistema, así como la asignación de los perfiles de cada usuario/aplicación.



## Perfiles

Los perfiles permiten definir las opciones de menú disponibles para cada tipo de usuario. El perfil de tipo MANAGER es obligatorio y debe ser utilizado para los usuarios que tienen acceso a la configuración del sistema. Para el resto de usuarios se deben crear otros perfiles.

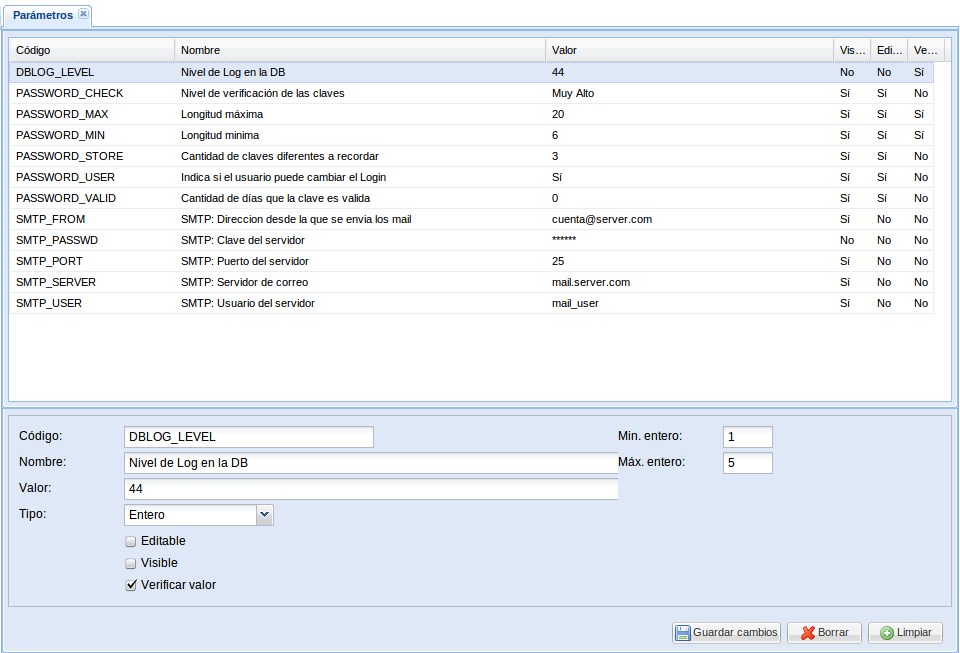


## Parámetros

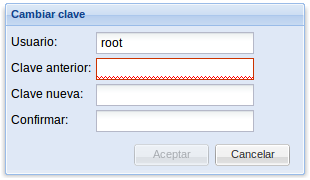
Mantenimiento de los parámetros de la aplicación. Se muestran dos vista de este mantenimiento según el perfil del usuario.

Para los perfiles MANAGER se muestra la siguiente ventana, donde se define el parámetro incluyendo su tipo, si es editable, si es visible y los chequeos de rango, así como los valores para los parámetros del tipo enumerativo.

Para los otros perfiles se muestra una vistan donde se puede listar la tabla de parámetros (solo los marcados como visibles) y editar el valor de los marcados como Editables, durante la edición se toma en cuenta el tipo del parámetro, así como los chequeos de rango definidos.



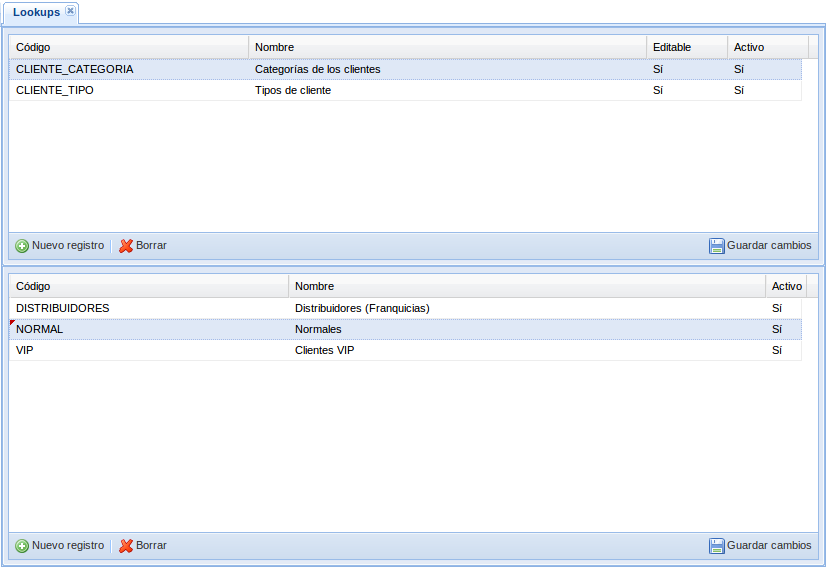
## Cambiar clave

Permite cambiar el usuario y la clave, tomando en cuenta el nivel de seguridad definido a través de los siguientes parámetros:

* *PASSWORD\_CHECK*: Nivel de seguridad para el control de usuarios
  + Ninguno: Las contraseña no tienen que cumplir ningún criterio y se almacenan sin encriptar
  + Bajo: Las contraseñas solo tienen que cumplir el criterio de tamaño mínimo y se almacenan encriptadas.
  + Medio: Además del anterior tienen que tener letras y números
  + Alto: Los anteriores más mayúsculas y minúsculas
  + Muy alto: Los anteriores más símbolos (Ej: @#~$%&)
* *PASSWORD\_MAX*: Longitud máxima de la clave
* *PASSWOR\_MIN*: Longitud mínima de la clave
* *PASSWORD\_STORE*: Cantidad máxima de claves diferentes a recordar (cambios de clave)
* *PASSWORD\_USER*: Indica si el usuario puede cambiar su Login
* *PASSWORD\_VALID*: Cantidad de días que la clave es válida, cero si las claves no caducan

## Lookups

Permite la definición de valores para hacer lookup (seleccionar valores del un combox) sin necesidad de definir varias tablas de DB para almacenar diferentes tipos de *lookup values*. Permite dos tipos de mantenimiento, el que los define y el que puede cambiar los valores.



## Leyenda del log

El framework permite la grabación de Log por cada usuario, tanto desde los Action como desde JavaScript. Esta funcionalidad es muy útil para aplicaciones que tengan que cumplir un nivel alto de protección de datos (como por ejemplo aplicaciones médicas). En base de datos los logs de usuario se almacenan en la tabla APPBS\_USUARIOS\_LOG, guardando información del usuario, fecha, nivel de log, tipo de log, identificador de sesión y hasta 5 campos libres. Para cada tipo de log se tiene una leyenda para visualizar el log de usuario de una forma legible (tabla APPBS\_USUARIOS\_LOG\_LEGEND). Cada campo libre se puede referenciar desde la leyenda como $<numero del campo>.

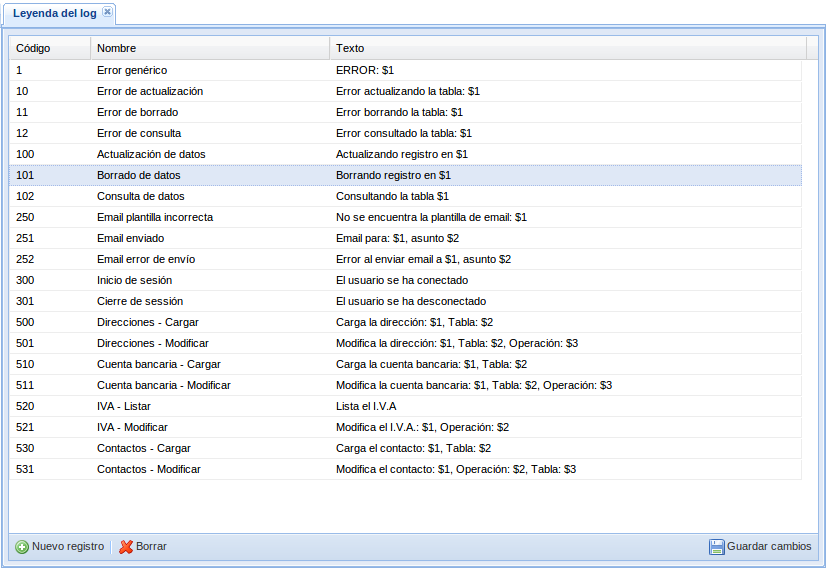
Ejemplo: 251; Email enviado; Email para: $1, Asunto: $2

Si se inserta un registro de log con la función de grabar Log:

***writeUserLog***(251, 2, ‘info@emp.com’,’Notificación de compra’);

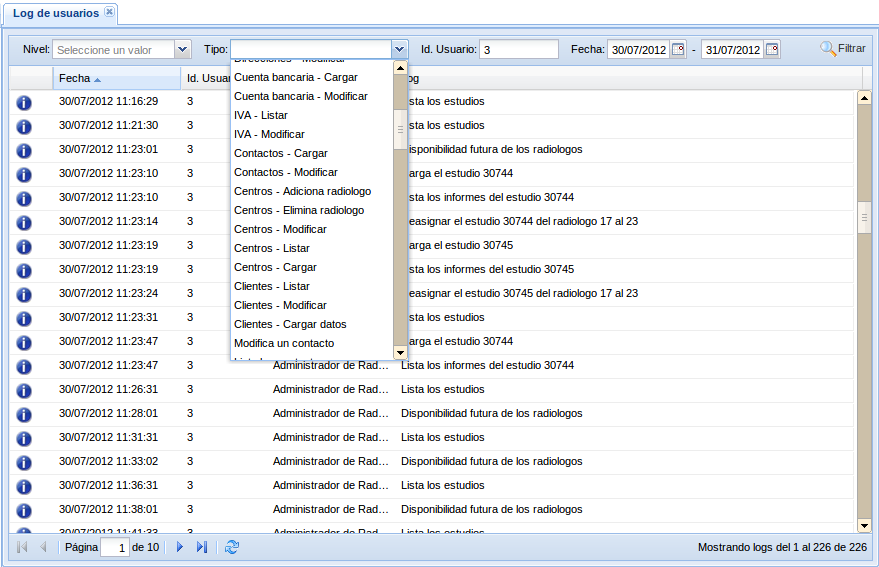
En la visualización del log de usuario se mostrará:

Email para: info@emp.com, Asunto: Notificación de compra



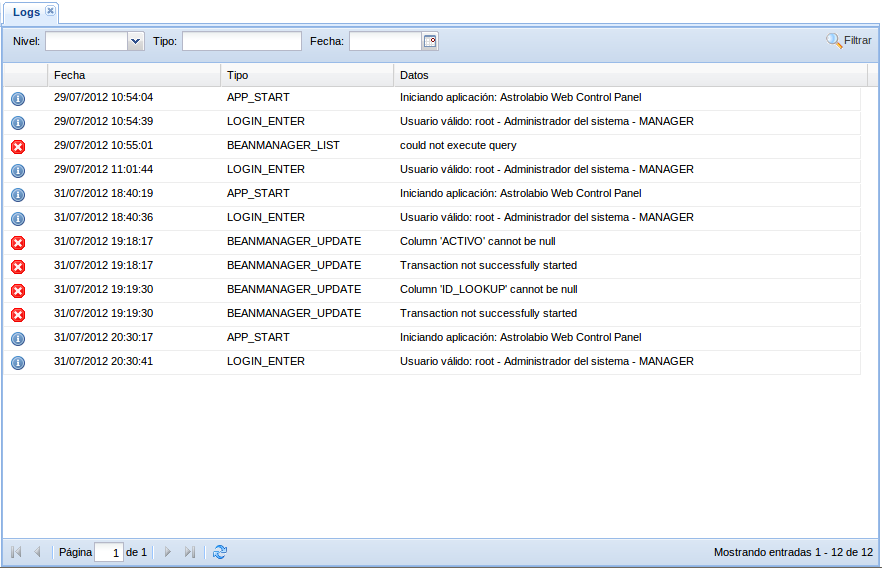
## Logs de usuario

Permite la visualización del Log almacenado por cada usuario, permitiendo el filtrado de los mismos.



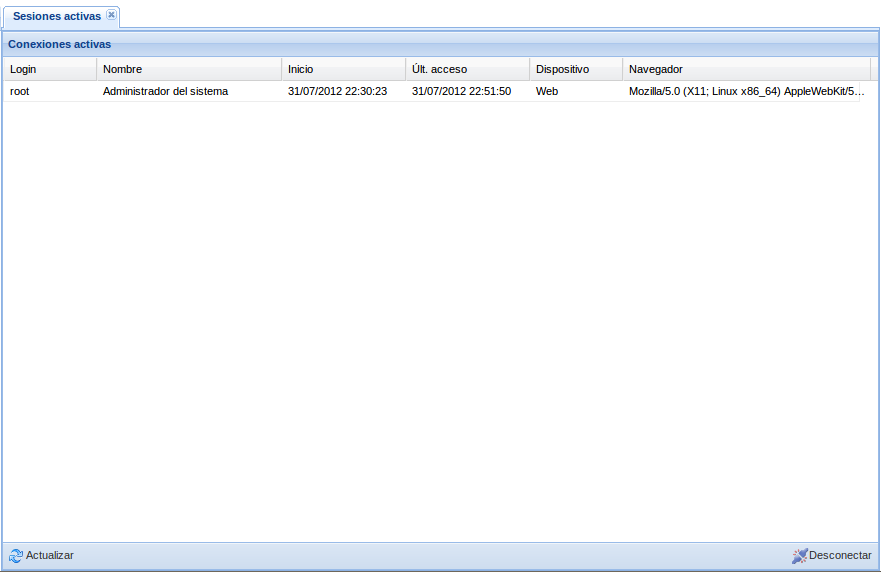
## Logs

Permite la visualización de la información de log registrada en la base de datos por la aplicación.



## Sesiones activas

Cuando el control de sesiones esta activo, se puede visualizar y controlar las sesiones abiertas en cada momento. Mostrando información en tiempo real de cada sesión, como el usuario, nombre, fecha de inicio de la sesión, último acceso, dispositivo desde el que accede (Navegador o Aplicación móvil), información del navegador. Además tiene la opción de cerrar una sesión.



# Mantenimientos incluidos en el Framework (Extendido)

Además de los mantenimientos básicos existen otra serie de tablas que también pueden incluirse en una aplicación, que tienen sus correspondientes mantenimientos

* Países, comunidades y provincias: Mantenimientos para gestionar las ubicaciones geopolíticas de Países, Comunidades y Provincias.
* Clientes, direcciones y contactos: Mantenimientos para gestionar clientes, direcciones de clientes y contactos de clientes y contactos de direcciones.
* I.V.A: Gestión del I.V.A.
* Bancos y cuentas bancarias: Gestión de bancos y cuentas bancarias.
* Idiomas y traducciones: Gestión de idiomas y traducciones de los campos que se quieran traducir.

# Estructura de la DB

A continuación se muestran dos gráficos con un resumen de la estructura de la base de datos.

