Manual Usuario gvHidra

Versión 4.4.33

Tabla de contenidos

I. Empezando con gvHidra 1
1. Empezando con gvHIDRA 3
1.1. Introducción 3
1.1.1. Acerca de este documento 3
1.1.2. ¿Qué es gvHIDRA? 3
1.1.3. Versiones 6
1.2. Requerimientos 6
1.2.1. Hardware 6
1.2.2. Software 6
1.3. Entendiendo el entorno 7
1.4. Instalación del entorno 9
1.4.1. ¿El entorno funciona correctamente? 10
1.5. Creando mi “hola mundo” en gvHidra 12
2. Conceptos técnicos de gvHidra 15
2.1. Arquitectura 15
2.1.1. Arquitectura por capas 15
2.1.2. Configuración 16
2.2. Aspecto visual 16
2.2.1. Anatomía de una ventana gvHidra 16
2.2.2. Modos de trabajo y ventanas gvHidra 17
2.2.3. Patrones de interfaz 18
2.3. Lógica de negocio 22
2.3.1. Acciones y operaciones 22
II. Elementos de gvHidra 28
3. Elementos básicos 31
3.1. Estructura de aplicación 31
3.1.1. Configuración estática: gvHidraConfig.xml 31
3.1.2. Configuración dinámica: AppMainWindow 36
3.1.3. Recomendaciones 37
3.2. Breve guía para crear una pantalla 41
3.2.1. Introducción 41
3.2.2. Genaro: Generación automática en gvHidra 41
3.2.3. Revisión de un mantenimiento 47
3.3. Menú de una aplicación 55
3.3.1. Funcionamiento del menú 55
3.3.2. Opciones predefinidas para el menu 60
3.3.3. Autenticación y autorización (módulos y roles) 61
3.4. Diseño de pantalla con Smarty/plugins 65
3.4.1. ¿Qué es un *template*? 65
3.4.2. Cómo realizar un *template* (tpl) 65
3.4.3. Documentación de los plugins 70
3.5. Código de la lógica de negocio 70
3.5.1. Operaciones y métodos virtuales 70
3.5.2. Uso del panel de búsqueda 80
3.5.3. Acciones no genéricas 84
3.5.4. Acciones de interfaz 87
3.6. Personalizando el estilo 90
3.6.1. CSS 90
3.6.2. Imágenes 91

3.6.3. Ficheros de configuración del custom 91
3.7. Tratamiento de tipos de datos 91
3.7.1. Características generales 92
3.7.2. Cadenas (gvHidraString) 93
3.7.3. Fechas 94
3.7.4. Números 99
3.7.5. Creación de nuevos tipos de datos 100
3.8. Listas de datos sencillas 101
3.8.1. Listas 101
3.8.2. Checkbox 107
3.9. Mensajes y Errores 108
3.9.1. Invocación desde código 110
3.9.2. Invocación como confirmación 110
3.10. Uso de datos por defecto 111
4. Elementos de pantalla avanzados 112
4.1. Patrones complejos 112
4.1.1. Maestro/Detalle 112
4.1.2. Árbol 120
4.2. Componentes complejos 126
4.2.1. Ventana de selección 126
4.2.2. Selector 130
4.3. Tratamiento de ficheros 131
4.3.1. Manejo de ficheros de imágenes 131
4.3.2. Manejo de ficheros de cualquier tipo 133
4.3.3. Importar datos a la BD desde fichero 134
4.4. Control de la Navegación. Saltando entre ventanas 135
4.4.1. Acumulador/Operador 136
4.4.2. Clave Ajena (salto modal) 139
4.5. Carga dinámica de clases 144
4.5.1. Introducción 144
4.5.2. Ejemplos de utilización 144
III. Complementos al desarrollo 146
5. Fuentes de datos 148
5.1. Conexiones BBDD 148
5.1.1. Bases de Datos accesibles por gvHidra 148
5.1.2. Acceso y conexión con la Base de Datos 148
5.1.3. Transacciones 153
5.1.4. Procedimientos almacenados 155
5.1.5. Recomendaciones en el uso de SQL 156
5.2. Web Services 157
5.2.1. Web Services en PHP 157
5.2.2. Generación del WSDL 159
5.2.3. Web Services en gvHIDRA 160
5.3. ORM 162
5.3.1. Introducción 163
5.3.2. Ejemplo: Propel ORM 163
6. Seguridad 180
6.1. Autenticación de usuarios 180
6.1.1. Introducción 180
6.1.2. Elección del método de Autenticación 181
6.1.3. Crear un nuevo método de Autenticación 181
6.2. Modulos y Roles 181
6.2.1. Introducción 181

6.2.2. Uso en el framework 184
6.3. Permisos 185
IV. Conceptos Avanzados 186
7. Conceptos Avanzados 188
7.1. Excepciones 188
7.1.1. gvHidraSQLException 188
7.1.2. gvHidraLockException 188
7.1.3. gvHidraPrepareException 188
7.1.4. gvHidraExecuteException 188
7.1.5. gvHidraFetchException 189
7.1.6. gvHidraNotInTransException 189
7.2. Log de Eventos 189
7.2.1. Introducción 189
7.2.2. Crear eventos en el log 190
7.2.3. Consulta del Log 191
7.3. Depurando mi aplicación 191
7.4. Envio de correo desde mi aplicación: IgepCorreo 191
7.4.1. Configuración 191
7.4.2. Métodos básicos 191
7.5. Creación de un custom propio para una aplicación de gvHIDRA 192
7.5.1. Pasos previos 192
7.5.2. Archivos CSS 193
7.5.3. Clases más usadas de bootstrap 193
7.5.4. Iconos mediante fuentes tipográficas (icon-fonts) 194
7.5.5. Tipos de clases 194
7.5.6. Cosas a tener en cuenta 195
7.5.7. Estilos generales de la aplicación 195
7.5.8. Personalización de Logos 196
7.5.9. Panel login 198
7.5.10. Pantalla de inicio 202
7.5.11. Acerca de 205
7.5.12. Ventanas de aviso 208
7.5.13. Partes genéricas de los paneles FIL LIS EDI 211
7.5.14. Partes genéricas de los paneles LIS EDI 217
7.5.15. Panel FIL 219
7.5.16. Panel EDI 221
7.5.17. Panel LIS 223
7.5.18. Panel maestro-detalle 226
7.5.19. Panel árbol 227
7.5.20. Solapas 229
7.5.21. Debugger 231
7.5.22. Ejemplos prácticos 232
8. Bitacora de cambios aplicados a gvHidra 240
8.1. Como migrar mis aplicaciones a otra versión de gvHidra 240
8.1.1. Versión 4.4.22 240
8.1.2. Versión 4.4.13 240
8.1.3. Versión 4.2.19 242
8.1.4. Versión 4.2.0 242
8.1.5. Versión 4.0.x 245
8.1.6. Versión 4.0.0 246
8.1.7. Versión 3.1.0 248
V. Apendices 251
A. FAQs, resolución de problemas comunes 253

A.1. ¿Cómo puedo recargar el maestro desde el detalle? 253
A.2. ¿Qué opciones de impresión/exportación ofrece el framework? 253
B. Pluggins, que son y como usarlos en mi aplicación 254
B.1. Documentación Plugins gvHidra 254
B.1.1. CWArbol 255
B.1.2. CWAreaTexto 256
B.1.3. CWBarra 257
B.1.4. CWBarraInfPanel 258
B.1.5. CWBarraSupPanel 259
B.1.6. CWBoton 259
B.1.7. CWBotonTooltip 262
B.1.8. CWCampoTexto 265
B.1.9. CWContendorPestanyas 267
B.1.10. CWContenedor 268
B.1.11. CWCheckBox 269
B.1.12. CWFicha 270
B.1.13. CWFichaEdicion 271
B.1.14. CWFila 272
B.1.15. CWImagen 273
B.1.16. CWLabel 274
B.1.17. CWMaps 276
B.1.18. CWInfoContenedor 277
B.1.19. CWInformation 277
B.1.20. CWLista 278
B.1.21. CWMarcoPanel 280
B.1.22. CWMenuLayer 281
B.1.23. CWPaginador 283
B.1.24. CWPanel 283
B.1.25. CWPantallaEntrada 286
B.1.26. CWPestanyas 286
B.1.27. CWRichAreaTexto 287
B.1.28. CWSelector 289
B.1.29. CWSlider 290
B.1.30. CWSolapa 291
B.1.31. CWTabla 292
B.1.32. CWTree 293
B.1.33. CWUpLoad 294
B.1.34. CWVentana 295
C. Listados Jasper en gvHIDRA 297
C.1. Integracion de Jasper en un mantenimiento 297
C.2. Depuración de errores al intentar ejecutar informes JasperReports en gvHidra 300
C.3. Recomendaciones y generalidades sobre integración gvHidra y Jasper 302
D. Instalación rápida del entorno 305
D.1. Introducción 305
D.2. Requisitos 305
D.3. Instalación entorno rápido 305

Lista de tablas

2.1. Conceptos comunes a todas las acciones genéricas 23
2.2. 23
3.1. Listado de Métodos 1 64
3.2. Listado de Métodos 2 64
4.1. Ficheros implicados en implementación 136
4.2. Ficheros implicados en implementación 140
5.1. Perfiles SGBD 149
6.1. Tabla de metodos 1 184
6.2. Tabla de metodos 2 184
7.1. Tabla de Excepciones 188
7.2. Tabla de clasificación de los eventos 190

Parte I. Empezando con gvHidra

Tabla de contenidos

1. Empezando con gvHIDRA 3
1.1. Introducción 3
1.1.1. Acerca de este documento 3
1.1.2. ¿Qué es gvHIDRA? 3
1.1.3. Versiones 6
1.2. Requerimientos 6
1.2.1. Hardware 6
1.2.2. Software 6
1.3. Entendiendo el entorno 7
1.4. Instalación del entorno 9
1.4.1. ¿El entorno funciona correctamente? 10
1.5. Creando mi “hola mundo” en gvHidra 12
2. Conceptos técnicos de gvHidra 15
2.1. Arquitectura 15
2.1.1. Arquitectura por capas 15
2.1.2. Configuración 16
2.2. Aspecto visual 16
2.2.1. Anatomía de una ventana gvHidra 16
2.2.2. Modos de trabajo y ventanas gvHidra 17
2.2.3. Patrones de interfaz 18
2.3. Lógica de negocio 22
2.3.1. Acciones y operaciones 22

Capítulo 1. Empezando con gvHIDRA

1.1. Introducción

1.1.1. Acerca de este documento

Este documento pretende ser un punto de referencia para todo aquel que se vaya a embarcar en el desarrollo de aplicaciones basadas en el *framework* gvHidra.

A lo largo de los diferentes capítulos, se van a ir mostrando las posibilidades del *framework*, las mejoras que aporta frente a otras opciones y, por supuesto, como instalar y ejecutar una primera aplicación básica. Con todo ello, se tendrá preparado un entorno de trabajo funcional para poder ir desarrollando y practicando los conceptos adquiridos a lo largo de este manual.

Este manual ha sido desarrollado por el equipo de gvHIDRA para favorecer y agilizar el acceso a cualquier usuario novel o avanzado en el desarrollo de aplicaciones basadas en este mismo *framework*.

Complementariamente existe una lista de distribución [http://listserv.gva.es/cgi-bin/mailman/listinfo/gvhidra_soporte] a la que cualquier desarrollador puede suscribirse y hacer llegar sus peticiones de ayuda o sugerencias para mejorar la herramienta.

Por otro lado existen unos servicios de archivo que permiten realizar búsquedas en el histórico de mensajes de la lista simulando el funcionamiento tipo "foro". Se puede acceder a las listas de gvHIDRA desde la interfaz de nabble [http://gvhidra.3754916.n2.nabble.com], obtener orígenes RSS e incluso escribir a la lista.

1.1.2. ¿Qué es gvHIDRA?

gvHidra son las iniciales de **G**eneralitat **V**alenciana: **H**erramienta **I**ntegral de **D**esarrollo **R**ápido de **A**plicaciones.

Si hay que dar una explicación rápida de qué es este proyecto, podemos decir que se trata de un entorno de trabajo (*framework*) para el desarrollo de aplicaciones de gestión en entornos web con PHP siguiendo una guía de estilo (una guía para unificar los criterios de aspecto y usabilidad en el proceso de desarrollo de aplicaciones).

Evidentemente, esto no responde de forma plena a la pregunta que da título a este punto. Por ello, vamos a explicar las motivaciones que nos llevaron a su creación y las características más importantes que cubre.

1.1.2.1. Un poco de historia

El Servicio de Organización e Informática (SOI) de la Conselleria de Infraestructuras y Transporte de la Generalitat Valenciana (CIT) ha trabajado tradicionalmente con la premisa de aumentar la productividad en base a entornos de trabajo que facilitan el desarrollo de aplicaciones. Concretamente, una de sus líneas de desarrollo, utilizaba unas plantillas en PowerBuilder que, en una arquitectura cliente servidor, reducían los tiempos de desarrollos resolviendo la parte general del problema.

Dentro de este marco, la CIT emprendió el proyecto gvPONTIS cuyo objetivo principal era migrar todos los sistemas a sistemas de código abierto. Entre otras facetas afectadas se encontraban los lenguajes de

programación (se seleccionaron PHP y Java), los SGBD (se decidió fomentar el uso de Postgresql), la arquitectura (pasar al desarrollo web/tres capas)...

Con todo ello, se decidió crear un proyecto en PHP que, basándose en la guía de estilo de las aplicaciones de la CIT, aportara las mismas ventajas que las anteriores plantillas de PowerBuilder: aumentar la productiva de nuestros desarrollos.

Pero claro, a todos esos requerimientos, teníamos que añadir las dificultades que generaba el nuevo ámbito de trabajo: el entorno web (HTML, Javacript,...), el enfoque OpenSource, ... Por tanto se decidió incorporar como requerimiento la simplificación del entorno de trabajo para un desarrollador.

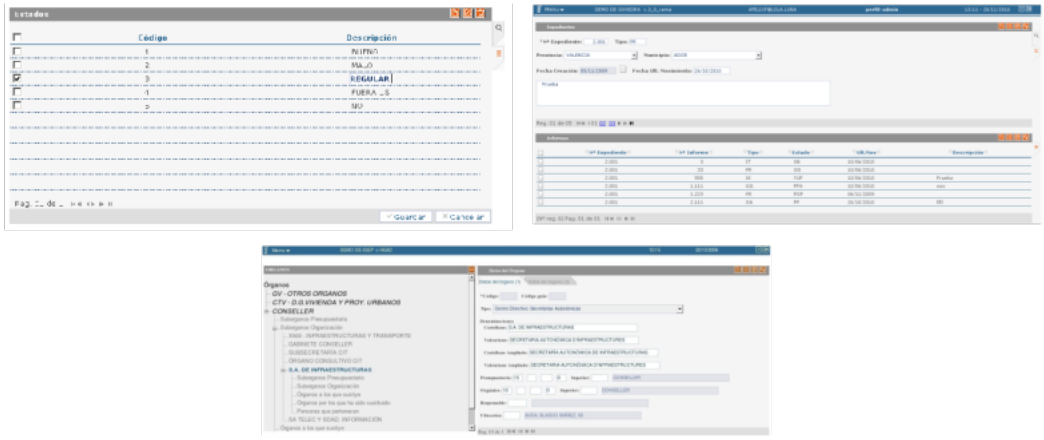
Con todo ello se creó un proyecto (igep: Implementación de la Guía de Estilo en Php) que componía el core del framework cuya primera versión estable salió el 16-11-2004. Este proyecto, al ser liberado con licencia GPL tomó la denominación gvHIDRA. A partir de este hito, empezaron a colaborar con el proyecto numerosas entidades públicas y privadas que nos ayudan a mantener el proyecto vivo y actualizado.

1.1.2.2. Pero ¿Qué es gvHIDRA?

Con lo expuesto en el punto anterior, quedan claras las premisas que nos movieron a plantearnos la creación de este proyecto pero, ¿Cómo hemos logrado unos objetivos tan ambiciosos? Bien, es dificil afirmar que los hemos alcanzado todos, pero creemos que buena parte de ellos quedan resueltos en el proyecto con las siguientes características del *framework*:

- **Patrones de interfaz**

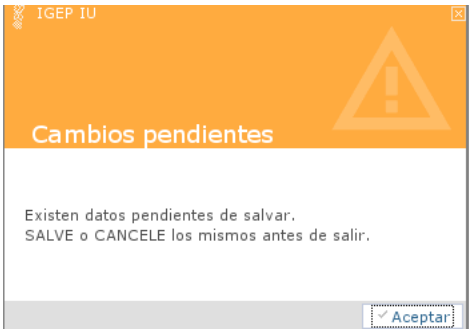
  Una de las tareas más costosas es en la definición de la interfaz con el usuario. Para facilitar el trabajo, se han definido una serie de patrones de básicos. Estos patrones definen la forma de representación de la información (formato tabular, registro,...) y la forma con la que interacciona el usuario con dicha información. Esto nos lleva a que con la selección de un patrón obtenemos el diseño global de la ventana.



  Ejemplos de patrón simple tabular, maestro detalle y árbol.

- **Componentes complejos**

  La experiencia acumulada nos dice que todas las aplicaciones necesitan de una serie de componentes “complejos”. Ventanas de selección (en PowerBuilder listas de valores), listas enlazadas, acciones de interfaz (en WEB tec. AJAX), mensajes de información,... El *framework* genera estos componentes simplificando su utilización en las aplicaciones.



- **Operaciones preprogramadas y parametrizables**

  Al igual que con los componentes, hemos advertido que cierta problemática se repite en todas las aplicaciones que desarrollamos. Por esa razón, la hemos generalizado y resuelto en el *framework*, siendo incorporada a las aplicaciones de forma transparente. Algunos ejemplos son:

  - Control de acceso concurrente: es importante que el garantizar la integridad de los cambios realizados por ello el *framework* incorpora de forma transparente un mecanismo de control.
  - CRUD: el *framework* genera las sentencias SQL necesarias para Crear, Leer, Actualizar y Borrar un registro.
  - Persistencia y validación de tipos: completando lo que nos ofrece el PHP, el *framework* incorpora objectos persistentes y validación de tipos de datos.

- **Soporte a diferentes SGBD**

  A través del proyecto PEAR::MDB2, el *framework* permite trabajar con diversos SGBD. Además, incorpora un capa de intermedia propia del *framework* que nos permite independizarnos de las diferentes interpretaciones del SQL que hace cada gestor (definición de límites, transacciones, ...).

- **Listados e informes**

  Uno de los mayores retos que hemos encontrado en el ámbito del proyecto fue generar informes de forma tan versatil como los datawindows de PowerBuilder. Gracias al proyecto jasperreports lo hemos conseguido. Apoyados por herramientas como el iReport, conseguimos listados e informes muy completos y en diferentes formatos.

- **Arquitectura MVC**

  El *framework* garantiza la arquitectura MVC en todos nuestros desarrollos forzando la separación de la lógica de negocio de la presentación mediante la distribución física de los ficheros fija.

- **Control de la vista**

  Como hemos comentado, uno de los objectivos principales de la herramienta es simplificar la labor del programador. Con esta premisa, hemos conseguido que un desarrollador de gvHIDRA realice una aplicación WEB sin necesidad de introducir ninguna línea de HTML o Javascript. La idea es que centre su trabajo únicamente en PHP, siendo así mucho más productivo.

- **Custom y temas**

  La herramienta está pensada para su despliegue en diferentes organizaciones, por ello, se distribuye en una arquitectura App/Custom/Core que permite modificar tanto el aspectos (CSS, imágenes, ...) como definir comportamientos propios de la organización.

- **Testing**

  Incorpora la herramienta PHPUnit para que se puedan realizar testeos sobre el código generado. También se incorporan consejos y reglas para poder realizar test automáticos son Selenium.

- **Autentificación, auditoría y depuración**

  Para poder incorporarse en diferentes organismos incorpora un mecanismo de validación extensible siendo capaz de acloplarse a cualquier sistema de validación a través de PHP. Dispone de herramientas para reliazar auditorías y depuración.

Como buen proyecto opensource, el proyecto siguen en constante evolución incorporando nuevas funcionalidades que nos permitan, ante todo, ser más productivos.

1.1.3. Versiones

Desde su liberación, el 16-11-2006, gvHIDRA ha seguido una estrategia de versiones con el fin de incorporar nuevas funcionalidades y resolver errores siendo la versión actual la 4.0.0 (rama 4.0.x). La numeración de las versiones corresponde al siguiente patrón: x.y.z Un cambio en el último número (z) implica corrección de errores/bugs, en este caso será invisible el cambio de versión en lo desarrollado a partir del framework. Si el cambio es en el segundo número (y), implica una mejora del framework o la adición de una nueva funcionalidad, esto sí puede provocar algún cambio en la programación de lo desarrollado a partir del framework. Por último, el cambio del primer número (x) sí que lleva una gran mejora en el framework o una nueva funcionalidad importante.

Recomendaciones:

- Trabajar siempre en la versión más reciente posible, para facilitar la resolución de errores, evitar usar una versión correspondiente a una rama no activa. Esto no significa que se descarten problemas de versiones anteriores, sino que si la solución implica cambios, éstos no se publicarán en ramas no activas.

1.2. Requerimientos

En este punto se explica las necesidades de un puesto cliente para poder ejecutar una aplicación realizada con gvHidra.

1.2.1. Hardware

Los requerimientos de hardware **minimos** recomendados para un correcto funcionamiento de una aplicación desarrollada con gvHidra son los siguientes:

- Pentium IV 2GHz o superior
- RAM 256 MB o superior

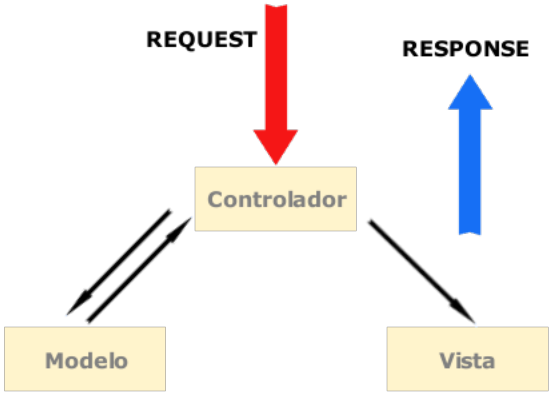
1.2.2. Software

En el caso de los requerimientos de software **minimos**

- Navegador mínimo: Firefox 3 o superior (actualizado el 17-01-2011) El navegador ha de permitir ventanas emergentes para la visualización de las ventanas de selección.
- Acrobat Reader (para visualizar listados)

1.3. Entendiendo el entorno

El *framework* gvHIDRA sigue una arquitectura MVC. Esta arquitectura tiene como objeto dividir en capas diferentes la lógica de negocio, la lógica de control y la presentación. Con ello conseguimos desarrollos más robustos y fuertes ante los cambios de requisitos. El siguiente diagrama muestra como funciona de forma esquemática la relación entre capas:



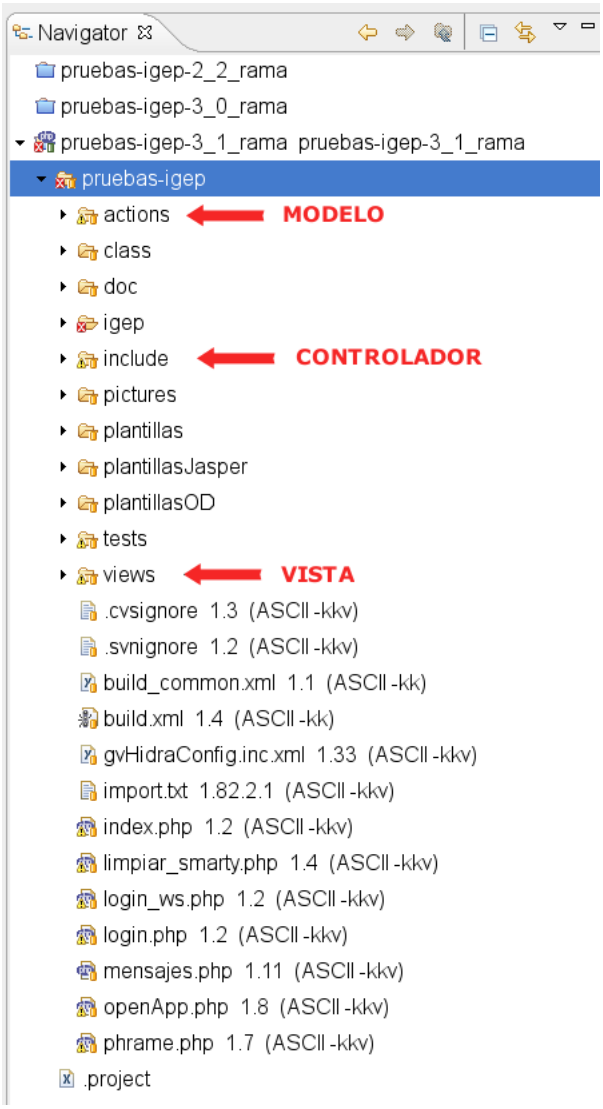
Esta división en capas se plasma físicamente dentro de una aplicación gvHIDRA. Concretamente, para hacer un mantenimiento, el programador tendrá que trabajar en las siguientes partes de la estructura:

- **actions**: corresponde al modelo y en él se alojan las clases manejadoras. Estas clases nos permiten acceder a los datos y realizar los tratamientos sobre ellos. Muchos de los tratamientos y comportamientos vienen heredados de forma que nuestra clase simplemente debe añadir el comportamiento específico.

- **include/mappings.php**: corresponde con el controlador. Basado en el proyecto Phrame, se compone de un fichero que nos permite asociar una acción (solicitud del usuario) con la clase que lo va a resolver.

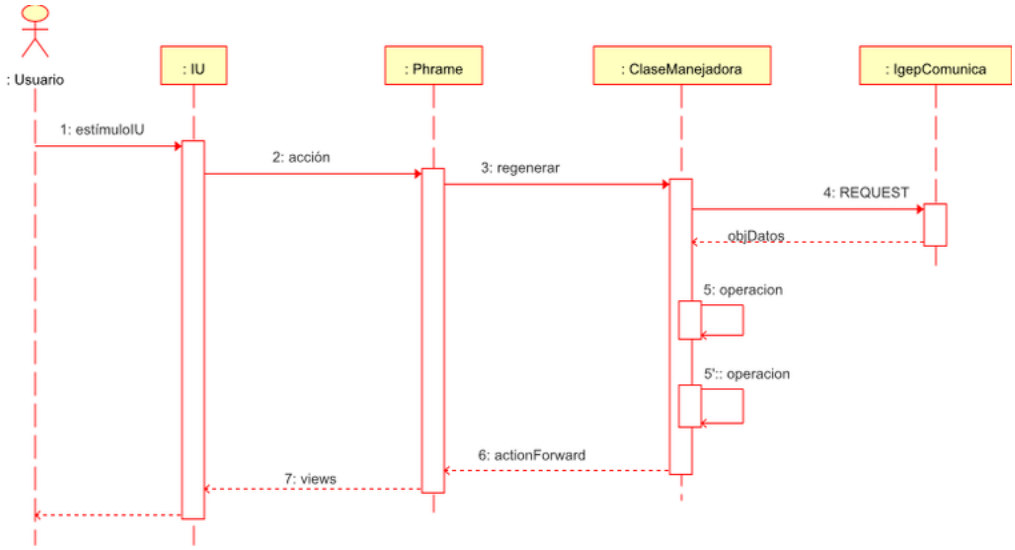
- **views**: corresponde con la vista. En este directorio encontraremos las vistas que designarán la pantalla a visualizarse.

Aquí tenemos un esquema donde podemos ver un ejemplo de aplicación con las capas que componen la arquitectura MVC.



A estas capas, falta añadir la interfaz con el usuario, las pantallas. Para la realización de las mismas, gvHIDRA hace uso de Smarty y de unos plugins propios. Estos fichero, se ubican en el directorio templates y contienen la definición de una ventana con todos sus componentes.

Una vez vista la estructura física de una aplicación, es conveniente ver los componentes internos del framework a través del flujo interno que provoca una petición de pantalla. En el siguiente diagrama de secuencia, hemos colocado algunos de los actores que intervienen en el tratamiento de una petición así como sus operaciones.



1. El flujo se inicia con la aparición de un estímulo de pantalla lanzado por el usuario.

2. Este estímulo es transmitido al controler (en nuestro caso phrame) en forma de acción. Ahora es phrame quien, consultado con el fichero de mapeos (fichero mappings.php) es capaz de conocer que clase es la encargada de gestionar la acción. Una vez conocida la clase, la "levanta" y le cede el control pasándole todos los datos de la petición.

3. La clase (conocida como clase manejadora) una vez tiene el control realiza varios pasos.

   - Reconecta de forma automática a la base de datos (en el caso de que exista).

   - Parsea el contenido de la petición (REQUEST) encapsulando en un objeto iterador que agrupa el contenido en matrices por operación.

   - Lanza las distintas operaciones. Estas operaciones se extienden con el comportamiento extra que añade el programador. Es decir, si lanzamos una acción de borrado, el *framework* realiza las operaciones necesarias para eliminar la tupla de la base de datos y el usuario tiene dos puntos de extensión opcional de dicho comportamiento: antes de borrar (métodos pre: para validaciones) y después de borrar (métodos post: para operaciones encadenadas).

4. Una vez finalizado todo el proceso, la clase manejadora devuelve un actionForward a phrame. Éste lo descompone y se localiza la views seleccionada.

5. Finalmente, el views recoge la información y, con la plantilla (fichero tpl de smarty) muestra la información en pantalla.

1.4. Instalación del entorno

Para ejecutar una aplicación en gvHIDRA, se tiene que disponer de un servidor web capaz de intepretar PHP. Estos son los requisitos técnicos:

- Servidor WEB. Puede ser cualquier servidor, aunque nosotros recomendamos Apache 2.X.

- PHP. Desde la versión 4.0.0 de gvHIDRA se puede utilizar el framework con versiones 5.3 o 5.4 de PHP.

- PEAR. Para el correcto funcionamiento del framework, se requieren una serie de librerías del proyecto PEAR con sus dependencias respectivas. Las requeridas son:
  - MDB2: capa de abstracción sobre la base de datos
  - MDB2_Driver_x: correspondiente al SGBD con el que vamos a trabajar.
  - PEAR: sistema básico de PEAR
  - Mail: para poder hacer uso de la clase de envio de correos del *framework*.
  - SOAP: si se quiere hacer uso de servidores de web services. Aunque PHP incorpora la la extensión SOAP, la librería PEAR::SOAP ofrece mecanismos para genarar WSDL de forma sencilla.

1.4.1. ¿El entorno funciona correctamente?

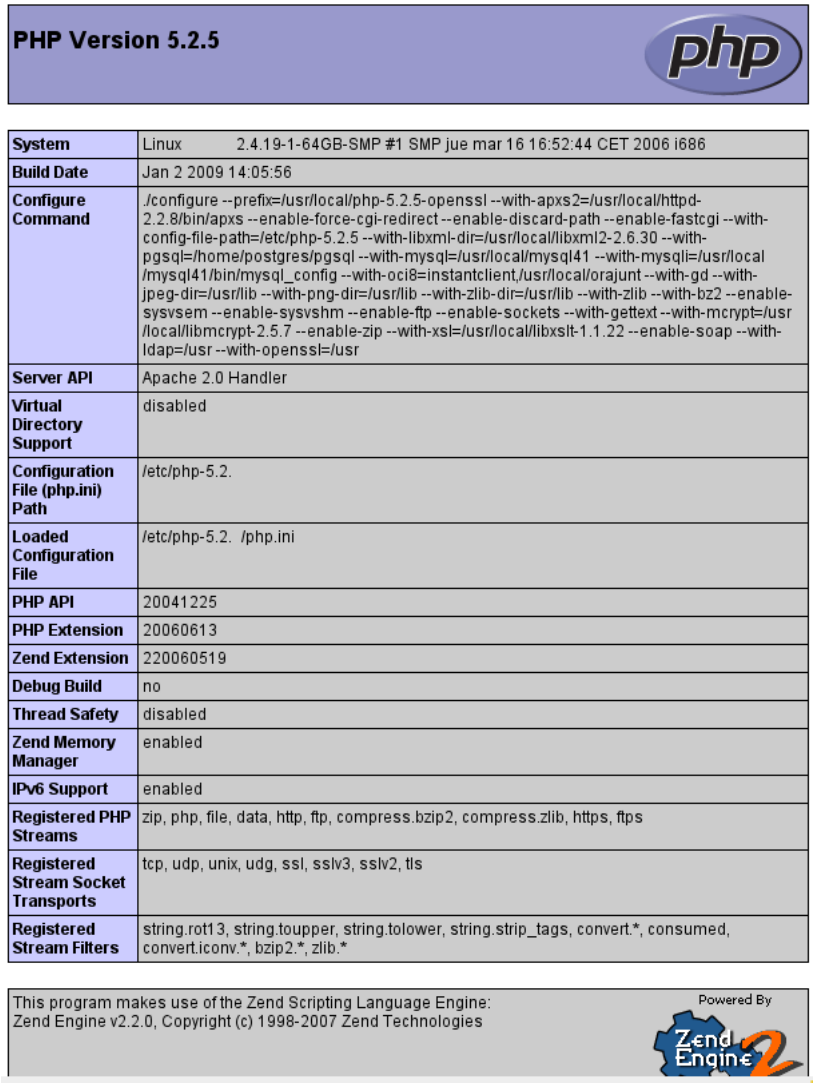
Lo primero que tenemos que comprobar es que el entorno de trabajo está funcionando correctamente. Para ello, vamos a ejecutar unos scripts que nos confirmarán que nuestro servidor está listo para albergar aplicaciones gvHIDRA.

1.4.1.1. ¿Funciona correctamente el PHP?

Creamos el siguiente script en la carpeta raiz del servidor web (generalmente, htdocs) con el nombre test1.php :

```
<?php
    phpinfo();
?>
```

Si todo ha ido bien, cuando accedamos a este script desde nuestro navegador tendremos algo parecido a la siguiente imagen.



PHP Version 5.2.5

System	Linux 2.4.19-1-64GB-SMP #1 SMP jue mar 16 16:52:44 CET 2006 i686
Build Date	Jan 2 2009 14:05:56
Configure Command	./configure --prefix=/usr/local/php-5.2.5-openssl --with-apxs2=/usr/local/httpd-2.2.8/bin/apxs --enable-force-cgi-redirect --enable-discard-path --enable-fastcgi --with-config-file-path=/etc/php-5.2.5 --with-libxml-dir=/usr/local/libxml2-2.6.30 --with-pgsql=/home/postgres/pgsql --with-mysql=/usr/local/mysql41 --with-mysqli=/usr/local/mysql41/bin/mysql_config --with-oci8=instantclient,/usr/local/orajunt --with-gd --with-jpeg-dir=/usr/lib --with-png-dir=/usr/lib --with-zlib-dir=/usr/lib --with-zlib --with-bz2 --enable-sysvsem --enable-sysvshm --enable-ftp --enable-sockets --with-gettext --with-mcrypt=/usr/local/libmcrypt-2.5.7 --enable-zip --with-xsl=/usr/local/libxslt-1.1.22 --enable-soap --with-ldap=/usr --with-openssl=/usr
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php-5.2.
Loaded Configuration File	/etc/php-5.2. /php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.2.0, Copyright (c) 1998-2007 Zend Technologies

Powered By Zend Engine 2

Podemos aprovechar para comproba que la versión de PHP es la correcta.

1.4.1.2. ¿Funciona la conexión al SGBD?

Bien, una vez hemos comprobado que tenemos una instalación correcta en lo relativo a PHP, vamos a comprobar que tenemos acceso a la base de datos. Para ello vamos crear un script que se conecte a nuestro SGBD. Para ello debemos copiar el siguiente código en el fichero test2.php:

```
<?php

include_once('MDB2.php');
include_once('PEAR.php');

$dsn= array(
  'phptype'  => 'xxxx',
  'username' => 'xxxx',
  'password' => 'xxxx',
  'hostspec' => 'xxxx',
```

```
  'port' => 'xxxx',
);

$options = array('portability' => MDB2_PORTABILITY_NONE,);
$res = MDB2::connect($dsn,$options);

if(PEAR::isError($res))
{
  print_r("Error:\n");
  print_r($res);
  die;
}

die("¡Eureka!");
?>
```

Ahora tenemos que completar la información con los datos necesarios para que el sistema localice la base de datos. Para ello, tenemos que completar la cadena de conexión con los siguientes valores:

- phptype: tipo de SGBD: oci8, oracle-thin, oracle-rac (para Oracle con TNS, explicitando conexión, o conectándonos a un cluster), pgsql (PostgreSQL), sqlsrv (SQL Server) o mysql.
- username: nombre de usuario para la conexión.
- password: password del usuario.
- hostspec: host del SGBD.
- database: nombre de la base de datos. No es necesario para Oracle.
- port: (opcional) puerto en el que está trabajando el SGBD. No es necesario si es el valor por defecto.

Con la información completa, sólo nos queda ubicar el fichero en una ubicación que sirva nuestro servidor web y acceder a la página con el navegador. Si todo ha funcionado correctamente debe aparecer en pantalla el texto "¡Eureka!". En caso contrario, conviene revisar los siguientes puntos:

- Instalación del paquete MDB2 y del driver MDB2 adecuado.
- Login y password del usuario.
- Acceso al SGBD, ¿tenemos acceso desde el servidor web al SGBD?

1.5. Creando mi “hola mundo” en gvHidra

Una vez tenemos el entorno preparado para el *framework*, vamos a probar una aplicación de pruebas. Entramos en la web del proyecto [http://www.gvhidra.org] y descargamos la última versión.

En este paquete ZIP, tenemos los siguientes componentes:

- Directorio doc: aquí tenemos la documentación relativa a la versión en formato HTML.
- Directorio doxy: ficheros generados con el doxy-gen que facilitan una navegación rápida entre las funciones del framework.

- Directorio igep: core del framework. Es el directorio que deberemos copiar en todas nuestras aplicaciones para poder trabajar con gvHIDRA.
- Directorio Phpdoc: información generada por el phpdoc.
- Guía rápida.txt: pasos mínimos de prueba del paquete.
- License.txt: referencia a la licencia GPL.
- Readme.txt: información básica del proyecto.

Una vez descargado, seguiremos los estos pasos:

1. Obtenemos el archivo plantilla-gvHidra.zip y lo descomprimimos en una carpeta del htdocs del servidor web.
2. Copiamos la carpeta igep que se encuentra en el paquete descargado en la carpeta creada en el paso anterior.

   Nota: puedes comprobar que la estructura es similar a la que hemos presentado en la imágen del punto 3.
3. Creamos carpeta templates_c y le damos permiso de escritura para el usuario Apache. Este directorio es el que utiliza el *framework* para compilar las plantillas.
4. Acceder a http://<servidor>/plantilla-gvHidra y validarse con el usuario 'invitado' y contraseña '1'.

Tras seguir estos pasos y, si todo ha ido bien, tenemos que obtener algo como esto:

Tras logarnos, entramos en la ventana principal de la aplicación que nos muestra las opciones de menu.

Enhorabuena ¡Ya tienes tu primera aplicación! Ahora puedes hacer uso del Genaro para crear los mantenimientos.

Capítulo 2. Conceptos técnicos de gvHidra

En este capítulo vamos a ver algunos conceptos teóricos que son necesarios antes de empezar a desarrollar aplicaciones con el framework. Con toda la información que comprende seremos capaces de entender e interpretar cada uno de los pasos que realicemos en capítulos posteriores.

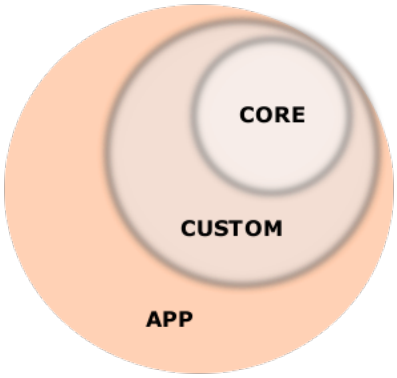
Se verán conceptos relativos a la arquitectura de las aplicaciones, a la interfaz con el usuario y a la lógica de negocio.

2.1. Arquitectura

gvHIDRA, como se ha comentado, es un proyecto opensource que se ha desarrollado para poder ser desplegado en entornos heterogéneos. Para ello su confirguración interna se ha organizado en una estructura de capas que permite redefinir los parámetros, crear clases específicas o cambiar el aspecto de cada una de nuestras aplicaciones.

2.1.1. Arquitectura por capas

Concretamente, el framework ofrece tres capas:



1. **core:** es la capa propia de los ficheros del framework. Está ubicada en el directorio igep y contiene ficheros de configuración propios del proyecto.
2. **custom:** destinado a las configuraciones propias de la entidad donde se despliega la aplicación. Generalmente, la organización donde se despliegue una aplicación tiene una serie de características propias:
   - aspecto: se puede configurar a partir de css e imágenes.
   - definición de listas o ventanas de selección generales (municipios, provincias, terceros, ...).
   - clases propias, tipos, ws, ...
   - configuraciones propias: conexiones a BBDD corporativas, acceso a datos comunes, formato de las fechas...
3. **app:** cada aplicación tiene una configuración propia:

- acceso a la BBDD propios de la aplicación.
- listas y ventanas de selección propias de la aplicación.
- configuraciones propias de la aplicación: nombre aplicación, versión, modo de debug/auditoria, comportamiento de la búsqueda, ...

Resumiendo, debemos tener en cuenta que hay varios niveles dentro de la arquitectura de una aplicación gvHIDRA y que en aras de la reutilización, conviene definir un custom propio de la organización en la que nos encontramos.

2.1.2. Configuración

En cuanto a la carga de la configuración, el framework la realiza en dos fases consecutivas siguiendo en cada una de ellas el orden (core/custom/app). Las dos fases se corresponden con:

- **carga estática:** esta fase carga los parámetros que estén ubicados en el fichero de configuración gvHidraConfig.inc.xml . Tenemos un fichero de este tipo en cada una de las capas, se cargarán en este orden:
  - igep/gvHidraConfig.inc.xml: fichero de configuración propio del framework. No debe ser modificado ya que contiene los parámetros propios del framework.
  - igep/custom/xxx/gvHidraConfig.inc.xml: fichero de configuración de la organización. Aquí se definirán las configuraciones propias de la organización.
  - gvHidraConfig.inc.xml:fichero de configuración propio de la aplicación.
- **carga dinámica:** esta fase, se realiza una vez acabada la anterior, con lo que machacará las configuraciones que se hayan realizado. Puede ser muy útil para fijar configuraciones de forma dinámica (dependiendo del entorno de trabajo), pero es peligroso, ya que puede ser difícil depurar un error. Los ficheros por capa se ejecutarán en el siguiente orden:
  - gep/actions/gvHidraMainWindow.php: fija la configuración dinámica del framework. No debe ser modificado.
  - igep/custom/xxx/ actions/CustomMainWindow.php: fija la configuración dinámica de la organización. En este fichero se deben definir las listas y ventanas de selección de la organización.
  - actions/principal/AppMainWindow.php: fija la configuración dinámica de la aplicación. En este fichero se deben definir las listas y ventanas de selección propias de la aplicación.

2.2. Aspecto visual

2.2.1. Anatomía de una ventana gvHidra

Antes de empezar a programar vamos a entender la anatomía de una ventana gvHidra.

Una ventana de gvHidra está dividida en 4 secciones:

- **Barra superior** (1)

  En esta barra aparece alineado a la izquierda el título de la ventana, y, alineado a la derecha aparecerán, si los hay, botones tooltip. Botones tooltip son botones que efectúan acciones relacionadas con la interfaz.

- **Contenedor** (2)

  Donde se ubicará el formulario con los datos y campos con los que trabajaremos.

- **Barra inferior** (3)

  En ella, alineado a la izquierda, se ubicará el paginador y, alineados a la derecha aparecerán los botones.

- **Contenedor de pestañas** (4)

  En esta zona irán apareciendo pestañas con las que podremos cambiar el modo de trabajo (búsqueda, listado, registro...)

2.2.2. Modos de trabajo y ventanas gvHidra

La forma de trabajar con las ventanas gvHidra se ha clasificado por modos de trabajo. Definiendo modo de trabajo como la forma de representación de la información con la que vamos a trabajar. Estos modos de trabajo se verán reflejados tanto en la zona *Contenedor* (2) del panel como en la de *Contenedor de pestañas* (4), la pestaña nos indicará el modo activo.

Los modos que tenemos son tres que se corresponden con las pestañas:

1. *Búsqueda o filtro:*
2. *Tabular o listado:*
3. *Registro o edición:*

Partiendo de todo lo comentado, tenemos dos formas de trabajar en gvHidra:

1. Dos modos de trabajo

   Con esto nos referimos al flujo de trabajo. En este caso partimos de una búsqueda y el resultado se muestra en una tabla o ficha, donde los datos ya son accesibles y se podrá operar sobre ellos.

Ej. Modo *búsqueda/tabla*

Ej. Modo *búsqueda/ficha*

2. Tres modos de trabajo

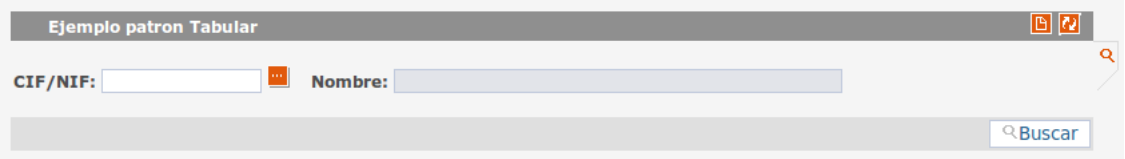
   En este caso es una combinación de los dos anteriores. Se parte de un panel de búsqueda, el resultado de dicha búsqueda se mostrará en un panel tabular, con la peculiaridad de que en este caso solamente se podrá efectuar el borrado de las tuplas que se seleccionen, para inserción o modificación se pasa a un panel modo ficha.

   Ej. Modo *búsqueda/tabla/ficha*

2.2.3. Patrones de interfaz

Una vez visto la estructura de la ventana y los modos de trabajo disponibles podemos pasar a explicar los diferentes patrones de interfaz que se pueden implementar con gvHidra.

Todos los patrones tienen en común el modo búsqueda:



[NOTA: Una buena práctica es utilizar el prefijo "fil" para denominar algunos parámetros o nombres de variables que hagan referencia al panel de búsqueda.]

Patrones:

1. Tabular [19]
2. Registro [19]
3. Tabular-Registro [20]
4. Maestro-detalle [20]
5. Árbol [22]

1. **Tabular**

   Se corresponde con la forma de trabajo dos modos de trabajo, panel de búsqueda y panel tabular donde se trabajará con los datos.

   [NOTA: Una buena práctica es utilizar el prefijo "lis" para denominar algunos parámetros o nombres de variables que hagan referencia al panel tabular.]

2. **Registro**

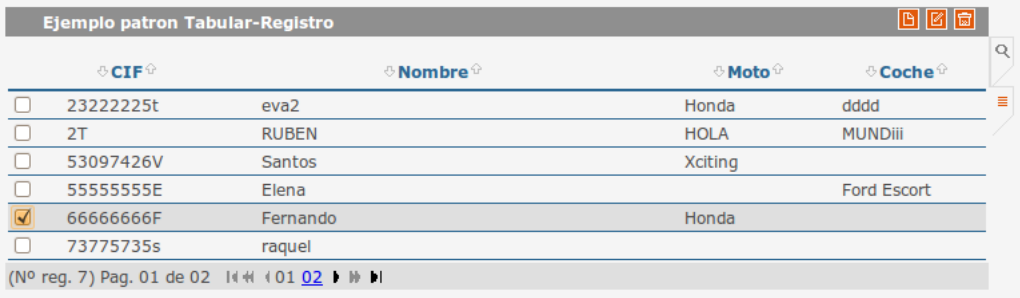
   Se corresponde con la forma de trabajo dos modos de trabajo, panel de búsqueda y panel registro donde se trabajará con los datos.

[NOTA: Una buena práctica es utilizar el prefijo "edi" para denominar algunos parámetros o nombres de variables que hagan referencia al panel registro.]

3. **Tabular-Registro**

Este caso se corresponde con la forma de trabajo de tres modos. Un panel de búsqueda, un panel tabular donde se tendrá el resultado de la búsqueda y un panel registro donde se podrán editar los campos de las tuplas seleccionadas en el panel tabular o insertar nuevas tuplas.

Modo tabular:



Modo registro:

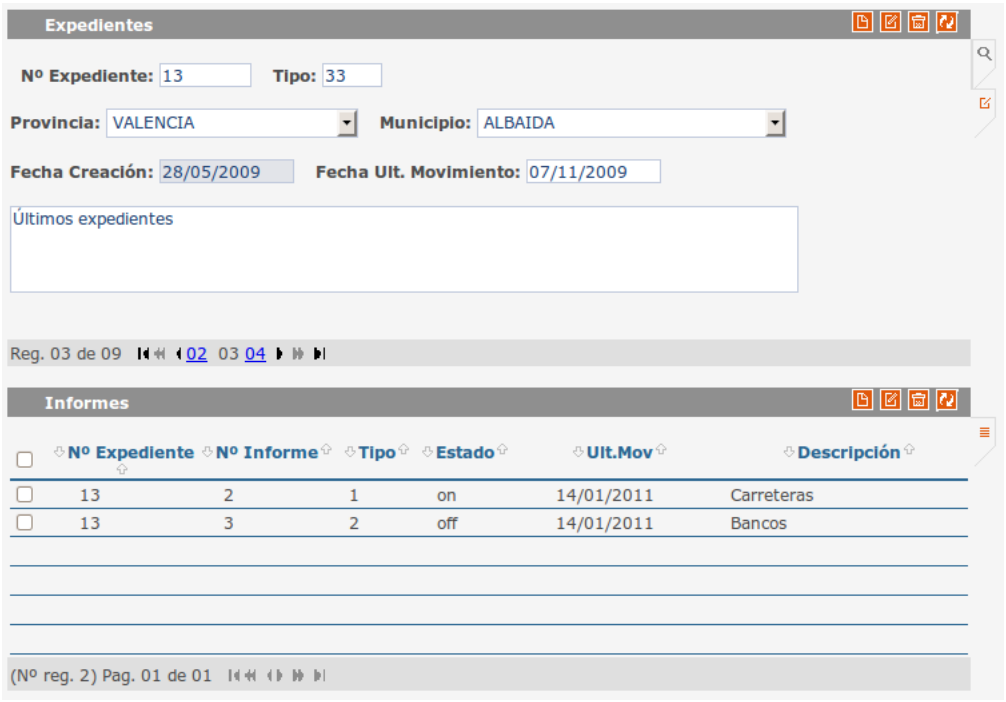
4. **Maestro-Detalle**

La parte del maestro será del tipo dos modos de trabajo, un panel búsqueda y un tabular o registro. En cambio en la parte correspondiente al detalle se puede plantear la forma de trabajar como dos modos (tener un sólo tabular o registro) o tres modos (tabular y registro), con la excepción de que en el detalle no tenemos búsqueda.

De esto podemos obtener diferentes combinaciones para un maestro-detalle:

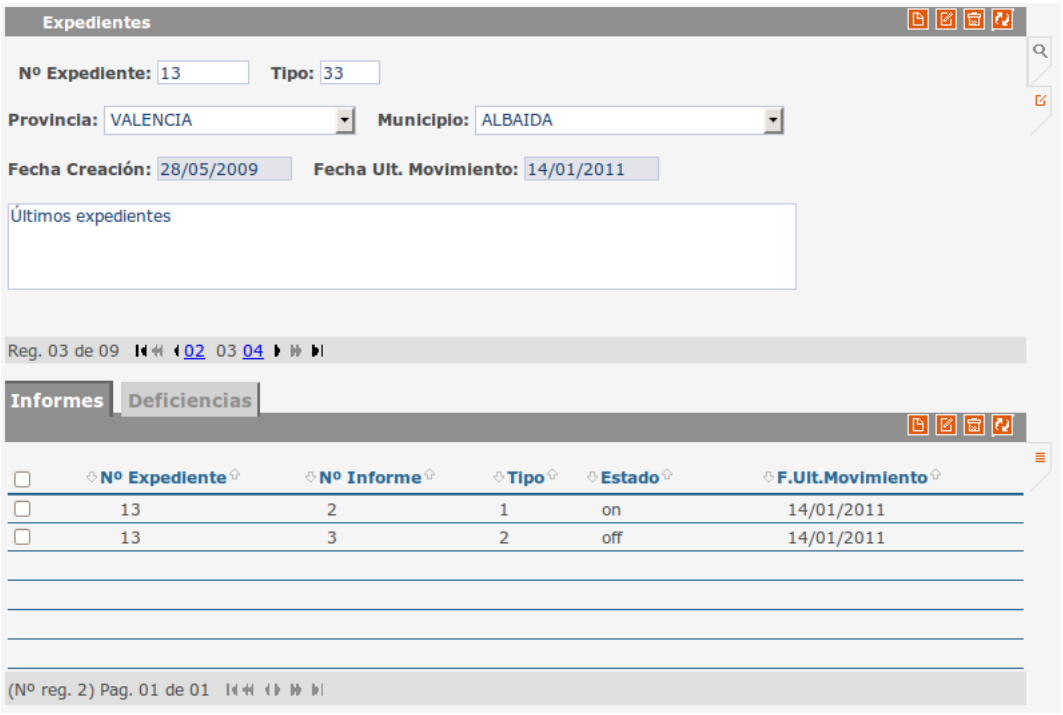
- Maestro tabular - Detalle tabular
- Maestro tabular - Detalle registro
- Maestro registro - Detalle tabular
- Maestro registro - Detalle registro
- Maestro tabular - Detalle tabular-registro
- Maestro registro - Detalle tabular-registro

Ej. Maestro registro - Detalle tabular:



Además de estos patrones para el maestro-detalle contamos con un patrón más complejo, el **Maestro – N Detalles**. Este patrón es una extensión de cualquiera de los indicados anteriormente, tendremos un maestro que tendrá varios detalles asociados. Estos detalles están accesibles mediante unas solapas que aparecerán en la cabecera de la zona del detalle.

Ej. Maestro registro - N Detalles:



5. **Árbol**

   El patrón árbol se compone de dos zonas. En la zona de la izquierda encontramos el árbol en sí, una jerarquía de menús, seleccionando cualquiera de ellos accederemos a su información que aparecerá representada en la zona de la derecha. Esta información podemos decidir representarla con un patrón Tabular, Registro o Tabular-Registro.

   Ej. Árbol:

2.3. Lógica de negocio

2.3.1. Acciones y operaciones

Acciones y operaciones son dos conceptos fundamentales que hay que conocer para desarrollar con el framework.

Una acción es el proceso que se inicia tras la solicitud de un usuario (un estímulo de la IU). Las acciones están divididas en dos grupos:

1. **Acciones genéricas**

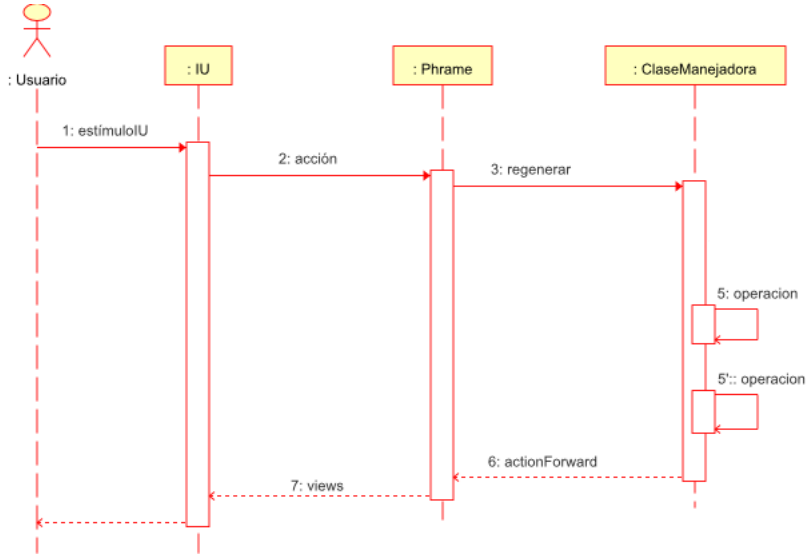
   Las acciones genéricas resuelven procesos comunes de un mantenimiento (altas, bajas, modificaciones, búsquedas...)

2. **Acciones particulares**

   Con las acciones particulares se podrán resolver necesidades concretas no resueltas por el framework.

Las **operaciones** son métodos internos del framework que serán llamados para cumplir lo que cada acción requiere, algunas de estas operaciones son públicas por lo tanto el programador podrá invocarlas desde las *acciones particulares*. El programador podrá modificar el comportamiento de por defecto de la operación sobreescribiendo con una serie de **métodos abstractos** (**virtuales**).

La siguiente imagen ilustra como se produce la comunicación entre las diferentes capas desde que un estímulo es recibido en la interfaz hasta que se resuelve. Podemos ver claramente la diferencia esencial entre acciones y operaciones.



Vamos a explicar un poco más en profundidad cada una de las acciones genéricas. Explicando tanto el objetivo global de la acción, los posibles retornos que produce, las operaciones que los componen y los métodos virtuales de cada una de estas operaciones.

Tabla 2.1. Conceptos comunes a todas las acciones genéricas

Métodos virtuales: Todas las acciones genéricas tienen una serie de métodos que el programador puede sobreescribir completando el comportamiento de la acción. Generalmente hay un método virtual “pre”, antes de la acción (inserción, modificación, borrado), y un método “post”, después de la acción.
actionForward: Acción de retorno de un método virtual. Puede tomar un valor por defecto asignado por el framework o el programador asignarle uno particular.

Tabla 2.2.

Valores de retorno de los métodos virtuales:
• **0**: Correcto, la ejecución continua. En este caso el actionForward se corresponde con la acción **gvHidraSuccess**. • **-1**: Ha habido un problema, el framework lanza un error si es problema interno de él, en cambio, si es un problema particular de la aplicación se ha de capturar el error y lanzar un mensaje propio (instrucción: *$this->showMessage('xxx');*) No se recargará la ventana. En este caso el *actionForward* se corresponde con la acción **gvHidraError**. • **actionForward**: Acción de retorno programada por el desarrollador para saltarse la ejecución de por defecto, provoca una recarga de la ventana.

Común a todas las acciones genéricas salvo a la de *Recargar*, esta acción no contempla una acción de retorno particular.

2.3.1.1. Acciones genéricas

1. **Iniciar ventana**

   Esta acción se ejecutará al iniciar la ventana.

   La acción contiene la siguiente operación:

   - *initWindow*: Genera todas las listas definidas para el panel y las carga en el objeto v_datosPreinsertar. Almacena el valor del módulo actual por si es la primera vez que se entra en una pantalla de dicho módulo. Su comportamiento se puede sobrecargar con el método virtual *preIniciarVentana*.

     preIniciarVentana: Método que permite realizar cualquier acción antes de que se cargue la ventana.

2. **Buscar**

   Realiza la consulta del panel basándose en los datos que hay en el panel filtro.

   La acción se compone de dos operaciones:

   - *buildQuery*: Método privado que recoge los datos del panel filtro y los añade al WHERE de la consulta del panel. Su comportamiento se puede sobrecargar con en el método virtual *preBuscar*.

     preBuscar: Método que permite realizar cualquier acción antes de que se realice la consulta.

   - *refreshSearch*: Método público que se encarga de realizar la consulta que se ha generado con el método anterior. Es importante saber, también, que este método puede ser llamado en cualquier momento para refrescar los datos de un panel, por ejemplo, después de una acción particular. Su comportamiento se puede sobrecargar con el método virtual *postBuscar*.

     postBuscar: Método que permite modificar los resultados obtenidos en la búsqueda.

     En el caso de que este método devuelva un 0 existe la posibilidad de tener un *actionForward* con la acción *gvHidraNoData* que mostrará un mensaje avisando que la consulta no ha devuelto datos, si se quiere particularizar este caso se sobrecargará la acción.

     Para los patrones simples Tabular-Registro, existe la posibilidad de añadir el *actionForward* con la acción *gvHidraSuccessOne* que permite agilizar el uso de la ventana cuando la búsqueda devuelve único registro pasando directamente al modo edición. Se ejecutarán los mismos métodos que al editar un registro.

3. **Editar**

   Esta acción genérica se ejecuta cuando estamos trabajando con tres modos. Como su propio nombre indica realiza la consulta de edición a partir de los datos seleccionados en el panel tabular.

   La acción se compone de dos operaciones:

   - *buildQueryEdit*: Método privado que recoge los datos del panel tabular con los que formará la WHERE de la consulta que nos devolverá los datos que se podrán editar. Su comportamiento se puede sobrecargar con el método virtual *preEditar*.

preEditar: Método que permite realizar cualquier acción antes de que se realice la consulta de edición.

- *refreshEdit*: Método público que se encarga de realizar la consulta que se ha generado con el método anterior, así obtener los datos actualizados. Es importante saber, también, que este método puede ser llamado en cualquier momento para refrescar los datos de un panel, por ejemplo, después de una acción particular. Su comportamiento se puede sobrecargar con el método virtual *postEditar*.

  postEditar: Método que permite modificar los resultados obtenidos de la consulta.

  En el caso de que no se haya seleccionado ninguna tupla para editar, aparecerá un mensaje por defecto informándonos de la situación. Para poder sobrecargar este comportamiento existe un *actionForward* con la acción *gvHidraNoData* con el que podremos hacerlo.

4. **Recargar**

   Acción genérica que actúa en un patrón maestro-detalle. Se ejecutará cuando se efectúe un cambio de maestro, recargará los detalles correspondientes.

   La acción se compone de dos operaciones:

   - *buildQuery*Details: Crea/recupera la instancia del panel detalle activo (puede haber N detalles activos). Llama al método *refreshDetail* de esa instancia detalle. Su comportamiento se puede sobrecargar con el método virtual *preRecargar*.

     preRecargar: Método que permite realizar cualquier acción antes de efectuar la consulta del detalle.

   - *refreshDetail*: Método público que se encarga de realizar la consulta que se ha generado en el método anterior, obtener los datos del detalle a partir del maestro seleccionado. Su comportamiento se puede sobrecargar con el método virtual *postRecargar*. Es importante tener en cuenta que el programador puede llamar a este método en cualquier momento para refrescar los datos de un detalle, por ejemplo, después de una acción particular.

     postRecargar: Método que permite modificar los datos obtenidos de la consulta.

     Hay que recordar que los métodos virtuales de esta acción no tienen como retorno un actionForward programado por el desarrollador.

5. **Modificar**

   Esta acción genérica se ejecuta desde el modo de trabajo edición, cuando, tras haber editado unas tuplas, se pulsa el botón guardar.

   La acción se compone de dos operaciones:

   - *updateSelected*: Método privado que recoge los datos de pantalla y actualiza en la BD los datos modificados. Su comportamiento se puede sobrecargar con el método virtual *preModificar*.

     preModificar: Método que permite realizar cualquier acción antes de que se lleve a cabo la modificación.

   - *refreshEdit*: Método público que se encarga de actualizar los datos visualizados en el modo de trabajo edición. Su comportamiento se puede sobrecargar con el método virtual *postModificar*.

     postModificar: Método que permite utilizar los datos modificados en otras operaciones.

- *refreshSearch o refreshDetail*: Método público que se encarga de actualizar los datos después de la operación de actualización en el panel. Es importante tener en cuenta que esto implica que se lanzará el *postBuscar* o el *postRecargar*.

La acción modificar por defecto deja el foco en el modo edición, este comportamiento se puede cambiar. El cambio se realiza en el fichero *mappings.php*, en la entrada correspondiente a modificar, en la acción *gvHidraSuccess*, cambiar en la url el panel destino (p.ej. panel=listado)

6. **Borrar**

Esta acción genérica se dispara "típicamente" desde el modo de trabajo tabular para eliminar las tuplas seleccionadas.

La acción se compone de dos operaciones:

- *deleteSelected*: Recoge las tuplas seleccionadas del panel y las elimina. Esta operación se puede parametrizar haciendo uso de los métodos virtuales *preBorrar* y *postBorrar*.

  preBorrar: Método que permite realizar cualquier acción antes de que se lleve a cabo el borrado.

  postBorrar: Método que permite modificar los resultados obtenidos después del borrado.

- *refreshSearch o refreshDetail*: Método público que se encarga de actualizar los datos después de la operación de actualización en el panel. Es importante tener en cuenta que esto implica que se lanzará el *postBuscar* o el *postRecargar*.

7. **Insertar**

Esta acción genérica se dispara desde el modo de trabajo inserción cuando, tras haber introducido los datos se pulsa el botón guardar.

La acción se compone de dos operaciones:

- *insertData*: Recoge los datos de la pantalla y los inserta en la BD. Esta operación se puede parametrizar haciendo uso de los métodos virtuales *preInsertar* y *postInsertar*.

  preInsertar: Método que permite realizar cualquier acción antes de que se lleve a cabo la inserción.

  postInsertar: Método que permite utilizar los datos modificados en otras operaciones.

- *refreshSearch o refreshDetail*: Método público que se encarga de actualizar los datos después de la operación de actualización en el panel. Es importante tener en cuenta que esto implica que se lanzará el *postBuscar* o el *postRecargar*.

8. **Nuevo**

Esta acción genérica se invoca para preparar la presentación de datos en pantalla antes de una inserción.

La acción se compone de la siguiente operación:

- *nuevo*: Realizará las operaciones de preparación de los datos y visualización previos a la inserción. Esta operación se puede sobreescribir con el método virtual *preNuevo*.

  preNuevo: Método que permite sobrecargar la acción nuevo antes de ser lanzada. Por ejemplo, cuando tenemos algún campo que tiene un valor por defecto o es calculado a partir de los valores de otros campos.

Validaciones de las acciones genéricas: este tipo de acciones tienen implícitas una serie de validaciones que hay que tener en consideración. Concretamente se comprueba que existan datos para realizar las operaciones pertinentes (p.e. datos para insertar, borrar, o actualizar), y si no es así finaliza la acción. Este comportamiento se puede deshabilitar haciendo uso de los métodos disableCheckEmptyOnInsert, disableCheckEmptyOnUpdate o disableCheckEmptyOnDelete. Deshabilitar esta validación permitirá en esos casos acceder al método virtual pre donde se podrán añadir datos de forma artificial. Por ejemplo, en un preBorrar se podrían añadir filas para borrar cuando el usuario no ha seleccionado nada.

Parte II. Elementos de gvHidra

Tabla de contenidos

3. Elementos básicos 31
3.1. Estructura de aplicación 31
3.1.1. Configuración estática: gvHidraConfig.xml 31
3.1.2. Configuración dinámica: AppMainWindow 36
3.1.3. Recomendaciones 37
3.2. Breve guía para crear una pantalla 41
3.2.1. Introducción 41
3.2.2. Genaro: Generación automática en gvHidra 41
3.2.3. Revisión de un mantenimiento 47
3.3. Menú de una aplicación 55
3.3.1. Funcionamiento del menú 55
3.3.2. Opciones predefinidas para el menu 60
3.3.3. Autenticación y autorización (módulos y roles) 61
3.4. Diseño de pantalla con Smarty/plugins 65
3.4.1. ¿Qué es un *template*? 65
3.4.2. Cómo realizar un *template* (tpl) 65
3.4.3. Documentación de los plugins 70
3.5. Código de la lógica de negocio 70
3.5.1. Operaciones y métodos virtuales 70
3.5.2. Uso del panel de búsqueda 80
3.5.3. Acciones no genéricas 84
3.5.4. Acciones de interfaz 87
3.6. Personalizando el estilo 90
3.6.1. CSS 90
3.6.2. Imágenes 91
3.6.3. Ficheros de configuración del custom 91
3.7. Tratamiento de tipos de datos 91
3.7.1. Características generales 92
3.7.2. Cadenas (gvHidraString) 93
3.7.3. Fechas 94
3.7.4. Números 99
3.7.5. Creación de nuevos tipos de datos 100
3.8. Listas de datos sencillas 101
3.8.1. Listas 101
3.8.2. Checkbox 107
3.9. Mensajes y Errores 108
3.9.1. Invocación desde código 110
3.9.2. Invocación como confirmación 110
3.10. Uso de datos por defecto 111
4. Elementos de pantalla avanzados 112
4.1. Patrones complejos 112
4.1.1. Maestro/Detalle 112
4.1.2. Árbol 120
4.2. Componentes complejos 126
4.2.1. Ventana de selección 126
4.2.2. Selector 130
4.3. Tratamiento de ficheros 131
4.3.1. Manejo de ficheros de imágenes 131
4.3.2. Manejo de ficheros de cualquier tipo 133

4.3.3. Importar datos a la BD desde fichero 134
4.4. Control de la Navegación. Saltando entre ventanas 135
4.4.1. Acumulador/Operador 136
4.4.2. Clave Ajena (salto modal) 139
4.5. Carga dinámica de clases 144
4.5.1. Introducción 144
4.5.2. Ejemplos de utilización 144

Capítulo 3. Elementos básicos

3.1. Estructura de aplicación

gvHIDRA, a parte de proporcionar un core (igep) en el que se encuentran todas las funcionalidades extensibles del framework, proporciona una estructura inicial de proyecto. Para empezar cualquier aplicación con la herramienta, debemos bajar la plantilla inicial de proyecto correspondiente a la versión con la que vamos a trabajar. Esta plantilla se encuentra en el paquete de la propia versión en la ubicación plantilla-gvHidra.zip. Antes de descomprimir hay que advertir que los ficheros están codificados en ISO-8859-1 o ISO-8859-15, por lo que hay que tener precaución con usar algún editor que respete estas codificaciones.

En el interior de la plantilla, encontramos una estructura básica de aplicación a partir de la cual construiremos nuestra aplicación. De esta estructura destacamos:

- **gvHidraConfig.inc.xml**: donde añadir las conexiones a bases de datos, código de la aplicación y versión,...
- **actions/principal/AppMainWindow.php**: donde añadiremos las listas y ventanas de selección particulares, log, ...
- **mensajes.php**: donde añadir mensajes particulares de la aplicación.
- **include/menuModulos.xml**: define el menu principal de la aplicación. Hay que modificarlo con las acciones propias de la aplicación. También podemos usar ciertas opciones predefinidas.
- **include/menuAdministracion.xml**: definimos opciones de la 2ª columna de la pantalla principal. Siguen las mismas normas que en menuModulos.xml.
- **include/menuHerramientas.xml**: definimos opciones de la 3ª columna de la pantalla principal. Siguen las mismas normas que en menuModulos.xml.
- **include/include.php**: donde, conforme vayamos creando clases de negocio las iremos incluyendo. Los includes de ficheros "action/*" se pueden borrar.
- **include/mappings.php**: en la función ComponentesMap, excepto la llamada al constructor del padre, todo se puede borrar.
- **templates_c**: directorio de compilación de plantillas. El usuario web tiene que tener permisos de escritura.

A esta estructura tenemos que añadirle el core del framework correspondiente (directorio igep) que también se distribuye con la versión de gvHIDRA.

3.1.1. Configuración estática: gvHidraConfig.xml

La configuración básica de una aplicación gvHIDRA se concentra en los ficheros de configuración gvHidraConfig.xml. En este apartado explicaremos tanto la distribución de estos ficheros entre la arquitectura del framework como las posibilidades que ofrece. Finalmente, veremos un ejemplo de un fichero tal y como se utiliza en una aplicación.

Aunque en algunos apartados se detalla mucha información, lo realmente importante es entender el concepto general y la configuración mínima que tenemos que realizar para crear una aplicación (fichero a nivel de aplicación).

3.1.1.1. Introducción

La configuración básica de una aplicación gvHIDRA se concentra en los ficheros **gvHidraConfig.xml**, donde se ubican todas las posibilidades de ajustes y parametrización.

Los ficheros de configuración mencionados, aparecerán en tres niveles, a nivel del framework (igep), otro a nivel de custom (personalización de la organización) y finalmente, a nivel de aplicación. Por ejemplo, supongamos una aplicación realizada con gvHidra que se denomine "factur" y se encuentra dentro de la organización "tecnimap" los ficheros de configuración se ubicarán en las estructura de directorios como sigue:

- factur/igep/gvHidraConfig.inc.xml
- factur/igep/custom/tecnimap/gvHidraConfig.inc.xml
- factur/gvHidraConfig.inc.xml

La semántica de los ficheros XML vendrá determinada a través de la DTD embebida en ellos, para que a los encargados de editar y mantener esos ficheros (programadores y responsables de la aplicación correspondiente) les sea sencillo saber si los mismos están correctamente confeccionados. La finalidad de que existan distintos ficheros de configuración, es establecer una especialización jerárquica, que permita personalizar opciones de gvHidra a distintos niveles.

- Nivel de **framework** (en el ejemplo anterior, fichero factur/igep/gvHidraConfig.inc.xml): En este fichero se establecerán configuraciones generales del framework y OBLIGATORIAMENTE se establecerá el directorio de customización (customDirName).
- Nivel de **organización** (personalización o custom, en el ejemplo anterior es el fichero factur/igep/custom/tecnimap/gvHidraConfig.inc.xml): Es un fichero que incluirá elementos propios de la organización (DSNs de BDs, acciones particulares de validación, ...), además en este fichero podrá redefinirse cualquier valor establecido en el fichero a nivel de Framework.
- Nivel de **aplicación** (en el ejemplo anterior, fichero factur/gvHidraConfig.inc.xml): Utilizado para incluir configuración específica de una aplicación (DSNs, versión...), como último en el nivel jerárquico, a través de él se podrá redefinir cualquier valor establecido en el fichero a nivel de organización y de framework.

  Además, el fichero del nivel de aplicación puede redirigir a un fichero con otro nombre y en otra ubicación, del servidor. Con ello, podemos conseguir (por ejemplo) facilitarse tareas de configuración a los equipos de sistemas, incrementar la seguridad valores de configuracion fuera del document_root, etc. (ver más adelante propiedad extConfigDir)

Es decir, en última instancia, las opciones que prevalecen son las que aparecen en el fichero de configuración de la aplicación. Posteriormente y a través del objeto global (el singleton ConfigFramework) podremos leer y sobrescribir dinámicamente dichas opciones.

3.1.1.2. Parámetros

A continuación vamos a detallar cada una de las opciones disponibles para la confección de los ficheros. Muchas de ellas no se deberán modificar al iniciar una aplicación, pero convien que las conozcamos:

- **applicationName**: De carácter obligatorio, establece el nombre (código) de la aplicación. La asignación normal será en la ubicación correspondiente a la aplicación.
- **appVersion**: Versión de la aplicación. La asignación normal será en la ubicación correspondiente al nivel de aplicación.

- **customTitle**: Breve descripción de la aplicación (texto que aparece en la pantalla principal).
- **barTitle**: En la barra superior de color azul, a la izquierda de la fecha y hora, se ha reservado un pequeño espacio para poder incluir un texto personalizado. La asignación se realiza a través de este parámetro.
- **customDirName**: Establece el nombre del directorio de customización. En dicho directorio aparecen modificaciones o extensiones que van a ser comunes a una organización. Por ejemplo, las extensiones del Framework propias de la CIT (ventanas de selección, listas predefinidas, DSNs de consulta a BDs internas...) se ubicarán dentro del directorio custom p.e "cit.gva.es". La asignación de este valor será en el fichero a nivel de framework o de la aplicación, no pudiéndose hacer a nivel de la organización ni de forma dinámica usando el objeto global.

  Desde la versión 4.1 de gvHidra podemos cambiar la ubicación del directorio custom, permitiendo que evoluciones de forma independiente del núcleo y que sea compartido (si se desea) por todo el conjunto de aplicaciones gvHidra de una misma organización.

  La forma de fijar la ubicación del directorio custom es mediante el atributo **path**, si no se indica nada, se busca (por compatibilidad) dentro de la carpeta igep de la aplicación
- **temporalDir**: Permite indicar una carpeta para la creación de algunos temporales relacionados con la seguridad (fichero de sesión, ...). La asignación de este valor será sólo en el fichero a nivel de la aplicación, no pudiéndose hacer a nivel de framework ni de la organización ni de forma dinámica usando el objeto global.
- **templatesCompilationDir**: Establece el directorio que utilizará Smarty para compilar las TPL. Si se está desarrollando algún plugin del framework, es interesante cambiar la configuración para que cada "usuario" tenga su carpeta y no haya un efecto caché mientras se desarrolla.
- **reloadMappings**: Indica si se quiere recargar el fichero de mappings en cada petición. Por defecto está habilitado, aunque habria que deshabilitarlo para entornos donde el fichero de mappings no se modifica. Este parámetro se puede definir a cualquier nivel.
- **smartyCompileCheck**: Indica si smarty tiene que comprobar si se ha modificado alguna plantilla y en caso afirmativo recargarla. Se puede definir en cualquiera de los tres xml, aunque no de forma dinámica.
- **logSettings**: Establece el nivel de detalle de los registros de auditoría.
- **breadcrumb**: Mostrará la ubicación del módulo en el que nos encontramos en la pantalla principal.
- **extConfigDir**: Permite definir una ubicación externa del fichero de configuración de la aplicación. Útil para, por ejemplo, ubicar los ficheros de configuración fuera del document_root.

  Además, si se desea utilizar otro nombre para el fichero, puede fijarse mediante al atributo **fileName** de dicha etiqueta, por lo que pueden centralizarse (si se desea) dichos ficheros en un directorio del servidor y restringir el acceso a los responsables de sistemas.
- **queryMode**: Modo de interrogar las BDs, vease los filtros de búsqueda.
- **DSNZone**: Encerrada en esta etiqueta aparecerá toda la información relativa a las fuentes de datos. La etiqueta podrá aparecer en las tres ubicaciones, por ejemplo: a nivel de framework para establecer la configuración del DSN del log, a nivel de organización para incluir DSNs propios de la organización (listas y ventanas de selección propias, validaciones...) y finalmente, a nivel de aplicación, donde situaremos la información de conexión propia.
  - **dbDSN**: Agrupa los parámetros de conexión a un SGBD relacional. Tiene dos parámetros obligatorios: id que referenciará la conexión en la aplicación y en Genaro, y sgbd que indicará el SGBD al que nos conectamos (postgres,oci8,mysql,sqlsrv o sqlite). Las etiquetas están inspiradas en PEAR:MDB2.

- *dbHost*: Host o IP donde se encuentra el servicio. En el caso de conexiónes a Oracle mediante OCI (cuando sgbd vale oracle, oci, oci8) se utilizará para establecer la entrada en tnsnames. Para los otros casos de oracle (thin u oracle-thin para conexiones explícitas sin TNS y oracle-rac para conexión a Real Application Cluster de Oracle) debemos especificar dbHost, dbPort y dbDatabase.
- *dbPort*: Puerto donde escucha el SGBD.
- *dbDatabase*: Base de datos a la que nos conectamos.
- *dbUser*: Usuario.
- *dbPassword*: Contraseña.

- **wsDSN**: La fuente de datos es un servicio web.
  - *uriWSDL*: URI donde puede localizarse el código WSDL que define el servicio web SOAP.
  - *wsUser*:Si el WS requiere autenticación vía USER/PASS indica el Usuario
  - *wsPassword*: Si el WS requiere autenticación vía USER/PASS indica la contraseña
  - *wsCertificateFilePath*: Si el WS requiere autenticación vía certificado, indicamos ruta (relativa o absoluta) al fichero del certificado.
  - *wsKeyFilePath*: Si el WS requiere autenticación vía certificado, indicamos ruta (relativa o absoluta) al fichero key.
  - *wsPassPhrase*: Si el WS requiere autenticación vía certificado, indica la frase de paso del mismo (si la tiene)
  - *wsPAITraceIdApp*: Si el WS invocado es de la PAI (Proyecto eSIRCA) y requeire trazabilidad, se indica el identificador de la aplicación que realiza la invocación. El atributo opcional *enablePAI-Trace* puede fijarse a true/false para activar o desactivar esta opción.

- **propertyZone**:Se utiliza para definir pares clave (id) valor en el XML de forma que puedan cumplimentarse parámetros no contemplados en el XML y que sean necesarios para desarrollar la APP manteniendo la libertad de modificar sus valores desde este fichero de configuración.
- **startMsg**:Contenido del mensaje que se quiera mostrar en el arranque de la aplicación o ruta al fichero que lo contiene.

En el siguiente apartado, tenemos un ejemplo de configuración a nivel de aplicación. Es especialmente interesante reparar en la configuración de las diferentes conexiones.

3.1.1.3. Ejemplo a nivel aplicación.

Después de esta abalancha de parámetros, para comprender mejor su utilización vamos a ver un ejemplo real de un fichero a nivel de aplicación:

```
<gvHidraConfig>
 ...
   <applicationName>factur</applicationName>
   <appVersion>1.0.0</appVersion>
   <!-- si es a nivel de organizacion, puede estar definido en el del custom--
>
```

```
   <logSettings status='LOG_ALL' dnsRef='gvh_dsn_log' />

   <DSNZone>
      <dbDSN id='gvh_dsn_log' sgbd='postgres'>
         <dbHost>cualquiera.coput.gva.es</dbHost>
         <dbPort>5432</dbPort>
         <dbDatabase>saturno</dbDatabase>
         <dbUser>logUser</dbUser>
         <dbPassword>melogeo</dbPassword>
      </dbDSN>

      <dbDSN id='ora-tns' sgbd='oracle'>
         <dbHost>tns-entry</dbHost>
         <dbUser>usuario</dbUser>
         <dbPassword>pwd</dbPassword>
      </dbDSN>

      <dbDSN id='ora-thin' sgbd='oracle-thin'>
         <dbHost>bd.coput.gva.es</dbHost>
         <dbPort>1521</dbPort>
         <dbDatabase>sid</dbDatabase>
         <dbUser>usuario</dbUser>
         <dbPassword>pwd</dbPassword>
      </dbDSN>

      <wsDSN id='g_ws'>
         <wsUser>wsuser</wsUser>
         <wsPassword>wspwd</wsPassword>
      </wsDSN>
   </DSNZone>
...
</gvHidraConfig>
```

Como nota, debemos saber que podemos acceder con la clase ConfigFramework a los metodos get/set para acceder/modificar estas propiedades. Ejemplo:

```
$conf = ConfigFramework::getConfig();
$cod_apli = $conf->getApplicationName();
$conf->setCustomTitle('Tu título');
```

3.1.1.4. Recomendaciones

Hay que tener en cuenta que, dependiendo del entorno en el que estemos trabajando, puede que nos interese habilitar/deshabilitar ciertos parámetros de configuración. Típicamente, nos encontramos con el caso de entornos de producción, en estos casos interesa:

- *smartyCompileCheck*: en producción es interesante tener este valor a false para optimizar el rendimiento de nuestras aplicaciones. Con ellos aprovecharemos la cache de Smarty.
- *reloadMappings*: en producción es interesante tener este valor a false para evitar que recarge el catálogo de acciones en cada ejecución. Esto aumentará la velocidad de nuestras aplicaciones.
- *extConfigDir*: dependiendo de la configuración del entorno y los sistemas de despliegue, puede ser interesante ubicar el fichero de configuración de la aplicación en una ubicación diferente al raíz de la aplicación. Si es este el caso, debes utilizar esta propiedad.

3.1.2. Configuración dinámica: AppMainWindow

La clase AppMainWindow es propia de cada proyecto, y se crea para inicializar caracteristicas generales de la aplicación. Como se ha explicado en los puntos anteriores, supone entre otras cosas, la carga dinámica de parámetros de configuración. Este fichero no es obligatorio; aunque en la práctica podemos decir que en el 100% de las aplicaciones es necesario.

Es importante tener presente que, por su definición, sólo se ejecutará la primera vez que entremos en la aplicación. A continuación describimos algunas de las funcionalidades que nos permite utilizar.

3.1.2.1. Carga dinámica.

Es la principal funcionalidad de esta clase. Si no la definimos, los valores de configuración serán los definidos en los ficheros gvHidraConfig.inc.xml. Sin embargo, cuando necesitamos fijar alguno de estos parámetros en funcion de situaciones más complejas (información propia al usuario, al estado de la aplicación, al entorno de ejecución, ...), podemos usar los 'setters' de los parámetros para cambiar la configuración. Por ejemplo:

```
$conf = ConfigFramework::getConfig();

// aumentar nivel de log en desarrollo
if ($estoyEnDesarrollo()) {
    // Cuando estamos en desarrollo registramos todos los movimientos
    $conf->setLogStatus(LOG_ALL);
}

// mostrar cierta informacion solo para perfil informaticos
if (IgepSession::dameRol()=='R_INFORMATICOS')
    $conf->setCustomTitle('perfil-admin');
```

3.1.2.2. Acciones genéricas de la ventana principal: abrir, volver al panel principal o cerrar la aplicación.

Cuando queremos hacer algo al empezar, al volver a la ventana principal o terminar la aplicacion, la definición de esta clase nos permite acceder a estas acciones específicas. Vease el ejemplo en el punto "Código de la lógica de negocio". Para ello hay que añadir al mappings.php las siguientes líneas:

```
//Accion que se ejecuta al abrir la aplicacion
$this->_AddMapping('abrirAplicacion', 'AppMainWindow');
$this->_AddForward('abrirAplicacion', 'gvHidraOpenApp', 'index.php?view=igep/
views/aplicacion.php');
$this->_AddForward('abrirAplicacion', 'gvHidraCloseApp', 'index.php?view=igep/
views/gvHidraCloseApp.php');

//Accion que se ejecuta al cerrar
$this->_AddMapping('cerrarAplicacion', 'AppMainWindow');
$this->_AddForward('cerrarAplicacion', 'gvHidraCloseApp', 'index.php?
view=igep/views/gvHidraCloseApp.php');

//Al volver al panel principal
$this->_AddMapping('volverPrincipal', 'AppMainWindow');
$this->_AddForward('volverPrincipal', 'gvHidraOpenApp', 'index.php?view=igep/
views/aplicacion.php');
```

```
$this->_AddForward('volverPrincipal', 'gvHidraCloseApp', 'index.php?view=igep/
views/gvHidraCloseApp.php');
```

3.1.2.3. Definir listas y ventanas de selección

Para poder añadir listas (gvHidraList) o ventanas de seleccion (gvHidraSelecctionWindow) a nuestras aplicaciones necesitamos definirlas en el constructor de esta clase. Hay más información al respecto en el apartado de las listas

3.1.2.4. Información global

Si se usan informaciones globales que no se tienen disponibles a través del patrón singleton, podemos usar los métodos guardaVariableGlobal y dameVariableGlobal de IgepSession para manejarlas. El constructor de AppMainWindow es el sitio natural donde inicializar los valores, para luego poder recuperarlos en cualquier parte de la aplicación.

3.1.3. Recomendaciones

En este apartado vamos a citar algunas de las recomendaciones que, aunque no son de obligado cumplimiento, si que nos pueden ser de gran utilidad ya que vienen recogidas de la experiencia de los desarrolladores.

3.1.3.1. Antes de arrancar

3.1.3.1.1. Configuración para desarrollo

gvHIDRA utiliza una serie de proyectos por debajo para implementar la arquitectura MVC y representar la vista. Concretamente phrame y a Smarty. Estos dos proyectos tienen una serie de parámetros de configuración que interesa tener en cuenta a la hora de trabajar en una aplicación. Lógicamente no es lo mismo estar en un entorno de desarrollo que en un entorno de explotación, por ello conviene tener en cuenta cuando estamos en uno u otro para realizar la configuración adecuada.

Todo ello se resume en estas dos propiedades del fichero gvHidraConf.xml a nivel de aplicación:

- **reloadMappings**: esta propiedad que admite valores boolenos(true|false), indica si el framework debe recargar el mapeo de acciones en cada iteracción. Esto es conveniente en desarrollo, aunque ralentiza la ejecución. *En explotación esta propiedad debe estar a false.*
- **smartyCompileCheck**: esta propiedad que admite valores boolenos(true|false), indica si regeneramos la plantilla (tpl) en cada interacción. En fase de desarrollo, como cambiamos constantemente la pantalla, es conveniente que esté a true. *En explotación esta propiedad debe estar a false* para mejorar el rendimiento.

3.1.3.1.2. Codificación

El framework está en la codificación **ISO-8859-1 (Latin1)**, por lo que el IDE que utilicemos para programar, debemos configurarlo en esta codificación. Con ello evitaremos problemas con caracteres especiales. En el caso de las conexiones a BBDD, si utilizamos Oracle, debemos definir la variable de entorno NLS_LANG=Spanish_Spain.WE8ISO8859P. En el resto de SGBDs no es necesario hacer nada.

3.1.3.1.3. Separar clases manejadoras por módulos

Es una buena práctica, crear carpetas dentro de los directorios actions, views y plantillas con los nombres de los módulos de la aplicación. Con esta sencilla separación física de los ficheros, será más sencillo localizar la información de un caso de uso concreto. Un ejemplo de módulos podría ser:

- Mantenimiento expedientes
- Tablas maestras
- Listados
- ...

3.1.3.1.4. Manual de Usuario Guía de Estilo

Aunque no es obligatorio, siempre conviene tener una guía para horientar a nuestros usuarios en el uso de las aplicaciones. Esta opción, abre una ventana emergente donde se explica el funcionamiento general de los distintos elementos de las aplicaciones realizadas con gvHIDRA. Se recomienda colocar en el menú de herramientas o de administración.

```
<opcion titulo="Manual Guía de Estilo" descripcion="Manual de introducción a
 Guía de Estilo"
 url="igep/doc/manualIGEP/indice/indice.html" imagen="menu/24.gif"
/>
```

También hay disponible una guía rápida:

```
<opcion titulo="Guía Rápida" descripcion="Guía
      rápida uso de aplicaciones"
      url="igep/custom/cit.gva.es/doc/guiaRapida/
guiaRapidaUsoAplicacionesCIT.pdf"
      abrirVentana='true' imagen="menu/24.gif" />
```

3.1.3.1.5. Acerca de...

De forma automática, el framework ofrece la posibilidad de crear un menu Acerca de que muestra el acrónimo y el nombre de la aplicación así como sus versiones. Para mostrarlo debemos añadir la siguente opción de menú:

```
<opcion titulo="Acerca de..."
 url="about" imagen="menu/24.gif"
/>
```

Al pulsar sobre esta opción el usario se encontrará con una pantalla similar a esta:

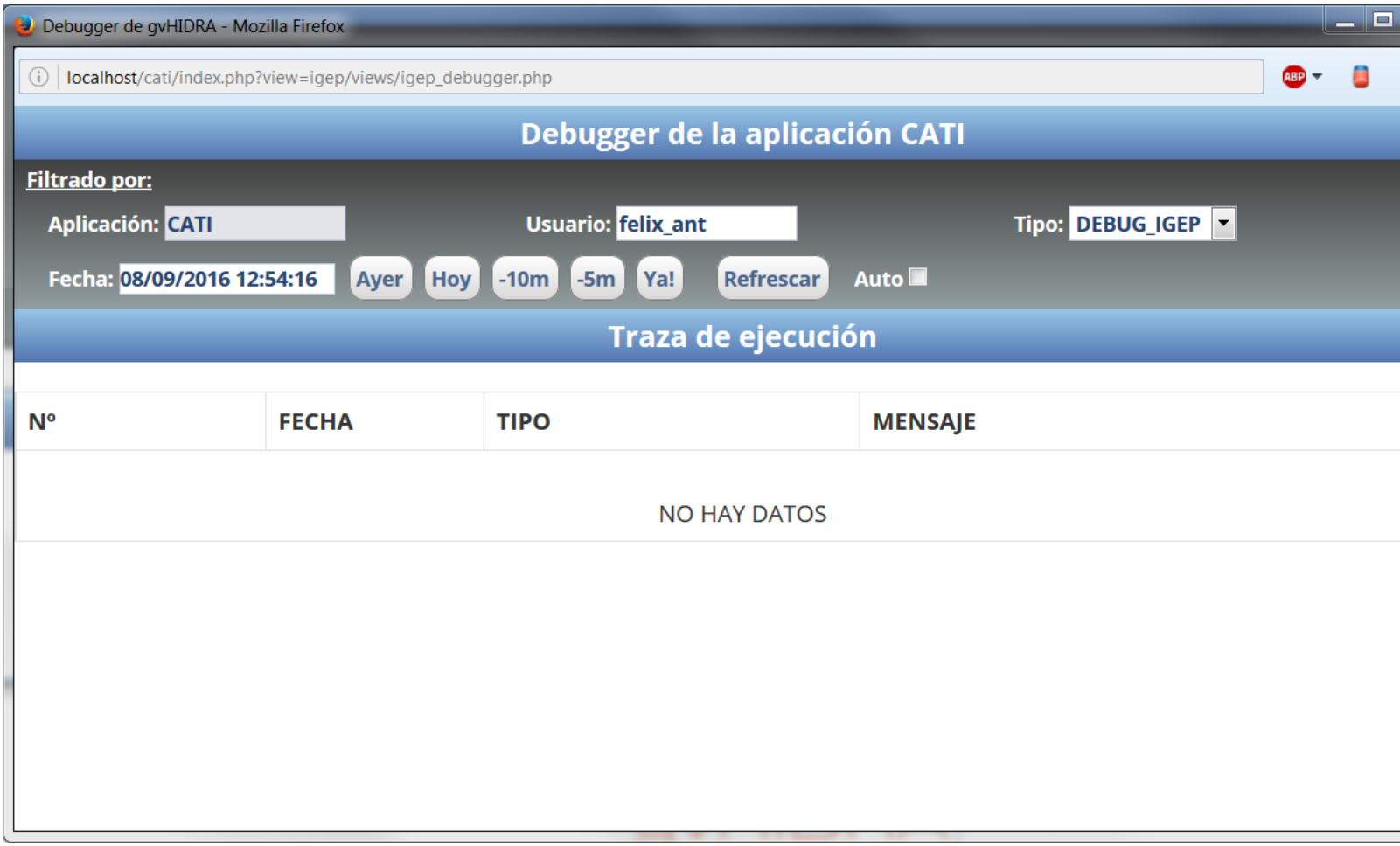
Para poder customizar esta ventana, el framework ofrece unas CSS que se pueden modificar en el fichero aplicacion.css.

3.1.3.2. Programando...

3.1.3.2.1. Consola de Log o Debug de la aplicación

Es una de las utilidades del framework básicas. Con el siguiente código se abre una ventana emergente para consultar la información generada por la aplicación. Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación"
        url="phrame.php?action=openDebugger" abrirVentana="true" imagen="menu/
mensajes.gif">
    <controlAcceso>
        ...
    </controlAcceso>
</opcion>
```

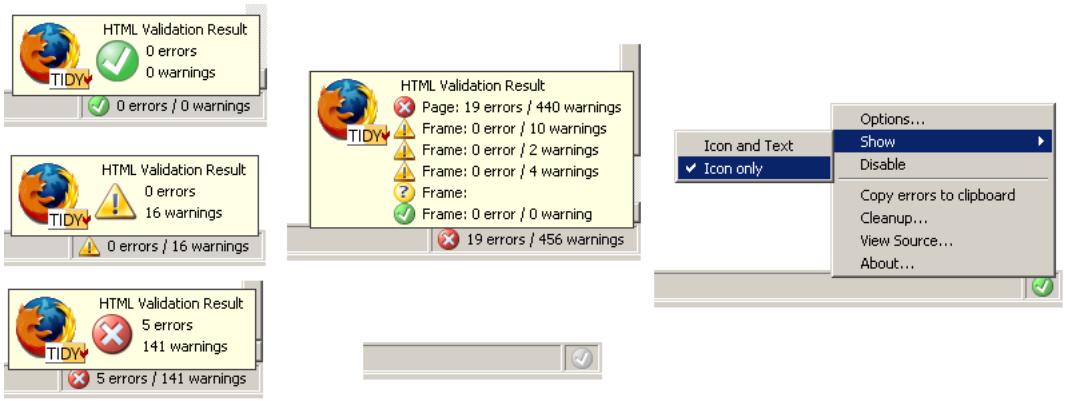


Más adelante explicaremos en profundidad el log del framework y sus posibilidades a modo de depuración y auditoría.

3.1.3.2.2. Herramientas para facilitar la depuración.

Además del propio debugger de la aplicación interesa instalarse alguna herramienta que nos permita ver el frame oculto del código HTML generado. Esto se debe a que, internamente, gvHIDRA trabaja con un frame que no es visible por el usuario, lo cual le permite, por ejemplo, realizar acciones sin recargar la página. Por todo ello, para la depuración, es fundamental instalar una herramienta que nos permita ver el contenido de dicho frame; en el caso de errores críticos, puede que sólo aparezcan en él.

Nuestra recomendación es utilizar el complemento de Firefox, HTML Validator; pero existen otras posibilidades. Aquí tenéis una imagen de dicho plugin de Firefox.



3.1.3.2.3. Problemas con el constructor de una clase

El framework tiene un sistema de persistencia de las clases manejadoras que permite, de forma transparente para el programador, mantener "viva" la instancia de una clase hasta que se cierre la ventana. Esto significa que, cuando se cierra una ventana y cuando se cierra la aplicación (a través de los iconos indicados), se borrarán de la sesión los datos de dicha clase. Es importante tener esto en cuenta porque si tenemos un error en el constructor, deberemos borrar la sesión antes de volver a ejecutar, ya que el constructor sólo se ejecuta la primera vez que entras a la ventana.

3.1.3.2.4. Proteger contraseñas

Se trata de una utilidad. Abre una ventana emergente donde podemos obtener el hash de una cadena (usados en servidores de web services). Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Proteger datos usando hash" descripcion="Proteger datos usando
 hash"
        url="igep/include/igep_utils/protectdata.php" abrirVentana="true"
 imagen="menu/seguridad.gif">
    <controlAcceso>
        ...
    </controlAcceso>
</opcion>
```

3.2. Breve guía para crear una pantalla

3.2.1. Introducción

A continuación vamos a explicar los pasos a seguir para crear una pantalla. Antes de llegar a este punto se tiene que haber seleccionado el tipo de pantalla que se quiere crear.

gvHidra está basado en la arquitectura MVC, por tanto al crear una pantalla se crean y modifican varios ficheros de la estructura:

- Fichero/s del **Modelo** (directorio *actions*): se creará un fichero que controla toda la lógica de negocio del panel.
- Fichero de la **Vista** (directorios *views* y *plantillas*): se crearán unos ficheros que, por un lado controlan la interfaz antes de presentar los datos, y por otro lado, la distribución de los componentes en la pantalla.
- Fichero de **Controlador** (ficheros *mappings.php* y *menuModulos.xml*): se modificarán estos ficheros para indicar qué acción se resuelve con qué clase.

Dentro del modelo, cabe tener en cuenta que el programador interacturá con el usuario a través de dos flujos: **flujo de entrada** y **flujo de salida**. El **flujo de entrada** se compone de la información que viene de la pantalla y se gestiona con:

- *Tablas de trabajo*: Son las tablas con las que el framework va a trabajar. Es decir, las tablas sobre las que vamos a realizar operaciones con los datos que recibamos.
- *Matching*: Este método nos sirve para indicarle al framework en que se traduce cierto campo de la pantalla. Se utiliza únicamente en los flujos de entrada: construcción de la where de la operación, del Insert, Update, ...

El **flujo de salida** se compone de la información a mostrar por pantalla y se gestiona a través de las consultas que definimos en el constructor de la clase:

- *SearchQuery*: es la consulta que se lanzará al ejecutar la acción "buscar".
- *EditQuery*: es la consulta que se lanzará al ejecutar la acción "editar".

Conviene refrescar estos dos conceptos una vez concluido el siguiente ejemplo.

3.2.2. Genaro: Generación automática en gvHidra

Desde las versiones 3.1.x, gvHidra cuenta con una herramienta de generación de código que nos permite generar de forma sencilla y rápida un mantenimiento. Esta herramienta, Genaro, a través de 5 parámetros básicos, se conecta a la base de datos y construye una ventana funcional. Los mantenimientos tienen la lógica necesaria para realizar la gestión de búsqueda, altas, bajas y modificaciones al 100%; pero no tienen lógica específica de la aplicación (validaciones específicas,...). Por esta razón, debe entenderse como un punto de partida para nuestras ventanas o prototipo.

3.2.2.1. ¿Qué genera?

La versión actual del genaro realiza por nosotros la generación de los siguientes ficheros:

- actions: genera las clases manejadoras (1 o N dependiendo del patrón). Estas clases tienen las consultas, las asociaciones con los campos, las asociaciones entre los maestros y sus detalles (en el caso de estos patrones) y los tipos de datos (con las validaciones de los mismos). En las versiones más recientes,

permite parametrizar los campos indicando el tipo de componente (CampoTexto, Lista, AreaTexto,...), el tamaño, ...

- views: genera el fichero de la vista.
- plantillas: genara la plantilla tpl de la ventana.
- include: edita el fichero include para incluir en el proyecto las nuevas clases.
- mappings: edita el fichero de mapeos para asociar las acciones con la clase que las gestiona (las nuevas clases manejadoras).
- menuModulos.xml: edita el menu para que aparezca la nueva entrada. Agrupa esta entrada por el módulo.

3.2.2.2. ¿Cómo lo instalo?

En primer lugar tenemos que descargar el paquete de la web de gvHidra.

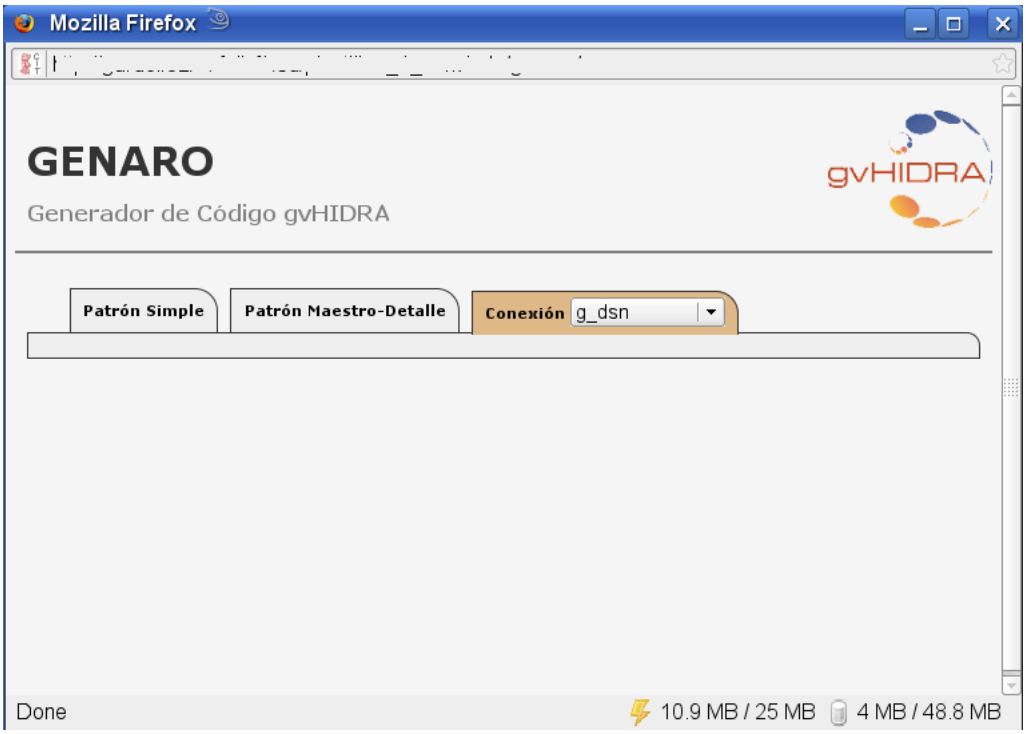
- Desde la página de gvHidra (www.gvhidra.org)

Una vez descargado, lo dejamos en el directorio include de nuestra aplicación gvHidra. Si no tenemos aun la estructura de la aplicación consultar el punto "Instalación del entorno" en este mismo manual.

NOTA: Hay que tener en cuenta que es necesario tener instalada la extensión php-xml en el servidor.

El genaro, recoge la información de las conexiones de nuestra aplicación, por lo que necesitamos que estén definidas en el fichero gvHidraConfig.inc.xml de la aplicación (ubicado en la raiz del proyecto).

Una vez definidas, debemos acceder a la herramienta http://<ubicacionAplicacion>/include/genaro. Por comodidad, se recomienda añadir esta entrada de menu en el menu de herramientas. Al acceder a esta url nos aparecerá el siguiente formulario.



Finalmente, para poder utilizar la herramienta tenemos que dar una serie de permisos de lectura/escritura ya que va crear/modificar ficheros en nuestra estructura. Por ello, se debe dar permisos de lectura/escritura al usuario del servidor web (generalmente el usuario de Apache) a todo nuestro proyecto.

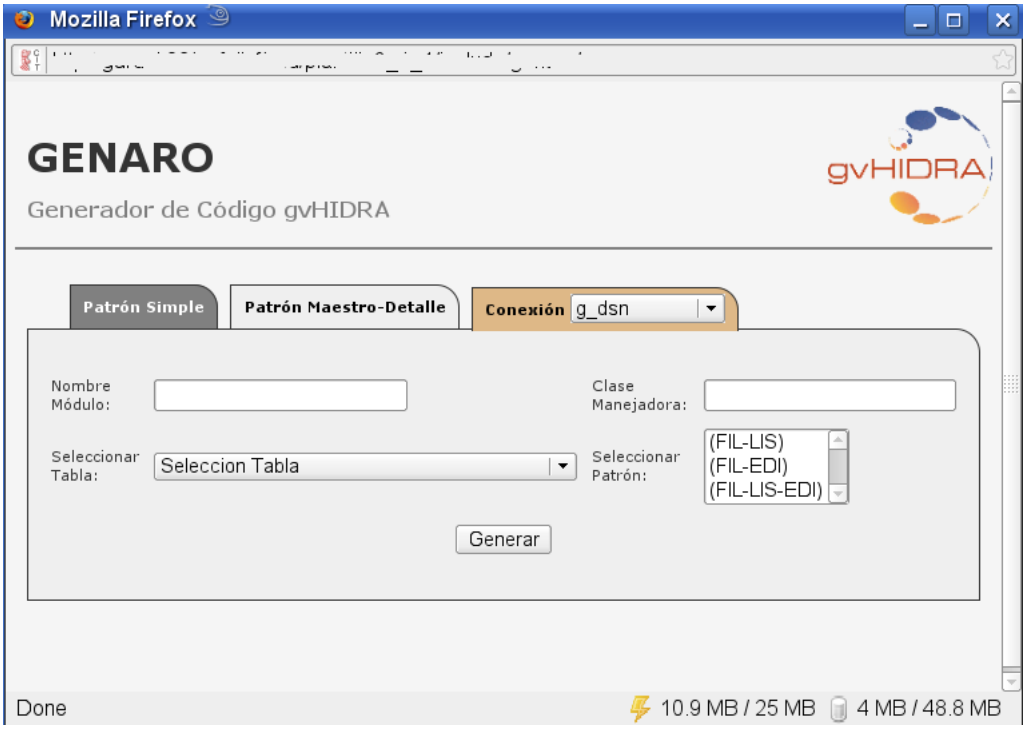
NOTA IMPORTANTE: Por seguridad, tanto el tema de permisos como el acceso a esta herramienta deben revisarse antes de sacar a producción. Se trata de una herramienta de desarrollo y SOLO debería estar disponible en dichos entornos.

3.2.2.3. ¿Cómo utilizarlo?

En primer lugar, tenemos que escoger la conexión de base de datos que queremos utilizar. Para ello, en el desplegable de la parte superior derecha (al lado de las solapa de Patrón Maestro detalle), seleccionamos el dsn deseado. Los ids que aparecen, son los que hemos configurado en nuestra aplicación.

Una vez seleccionada la conexión, debemos decidir si vamos a generar un patrón simple o un patrón maestro detalle. Vamos a revisar cada una de las ventanas.

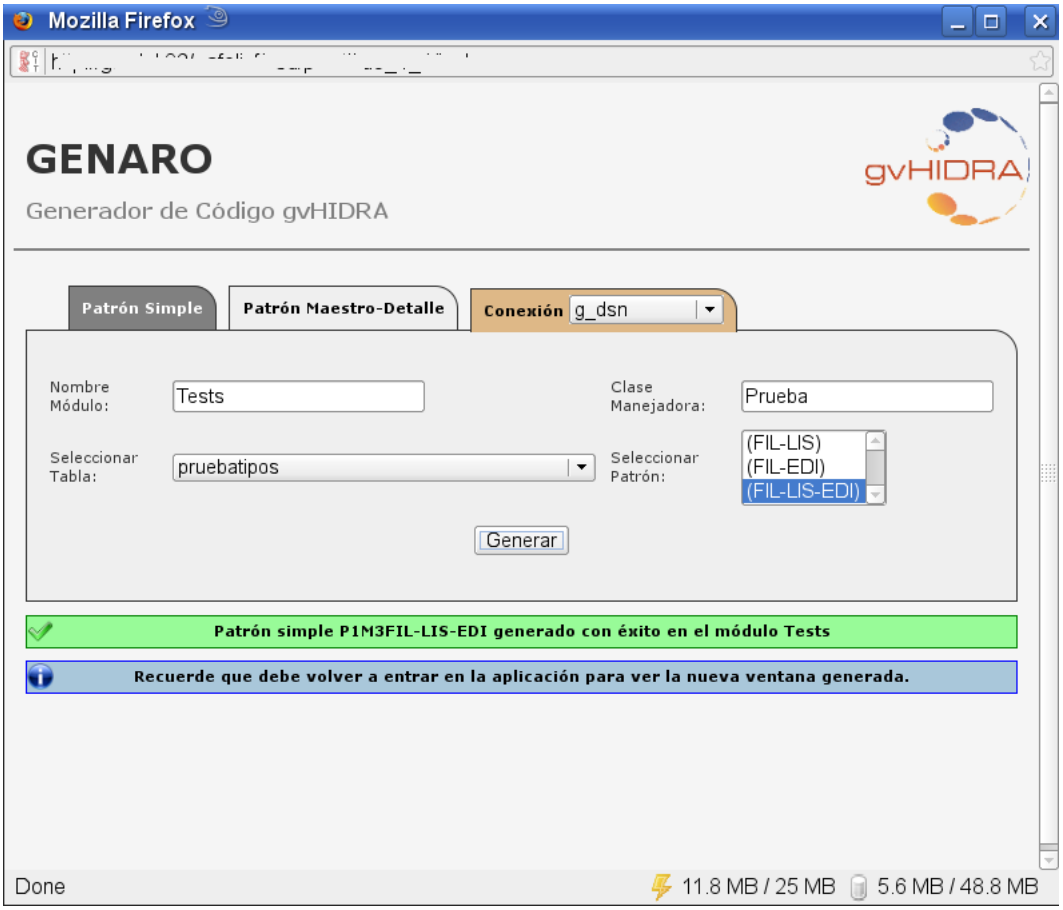
Para genarar un patrón simple tenemos la siguiente ventana



Debemos indicar los siguientes parámetros:

- Nombre del módulo: es un agrupador de mantenimientos. Nos permite agrupar los mantenimiento de nuestra aplicación de forma que sea más sencillo su manejo.
- Clase Manejadora: el nombre de la clase.
- Tabla de la bd: tabla sobre la que queremos realizar el mantenimiento.
- Patrón: seleccionamos la forma de visulización de la información (Tabular, Registro o Tabular-Registro).

Pulsamos generar y aparece la siguiente ventana:



Nota

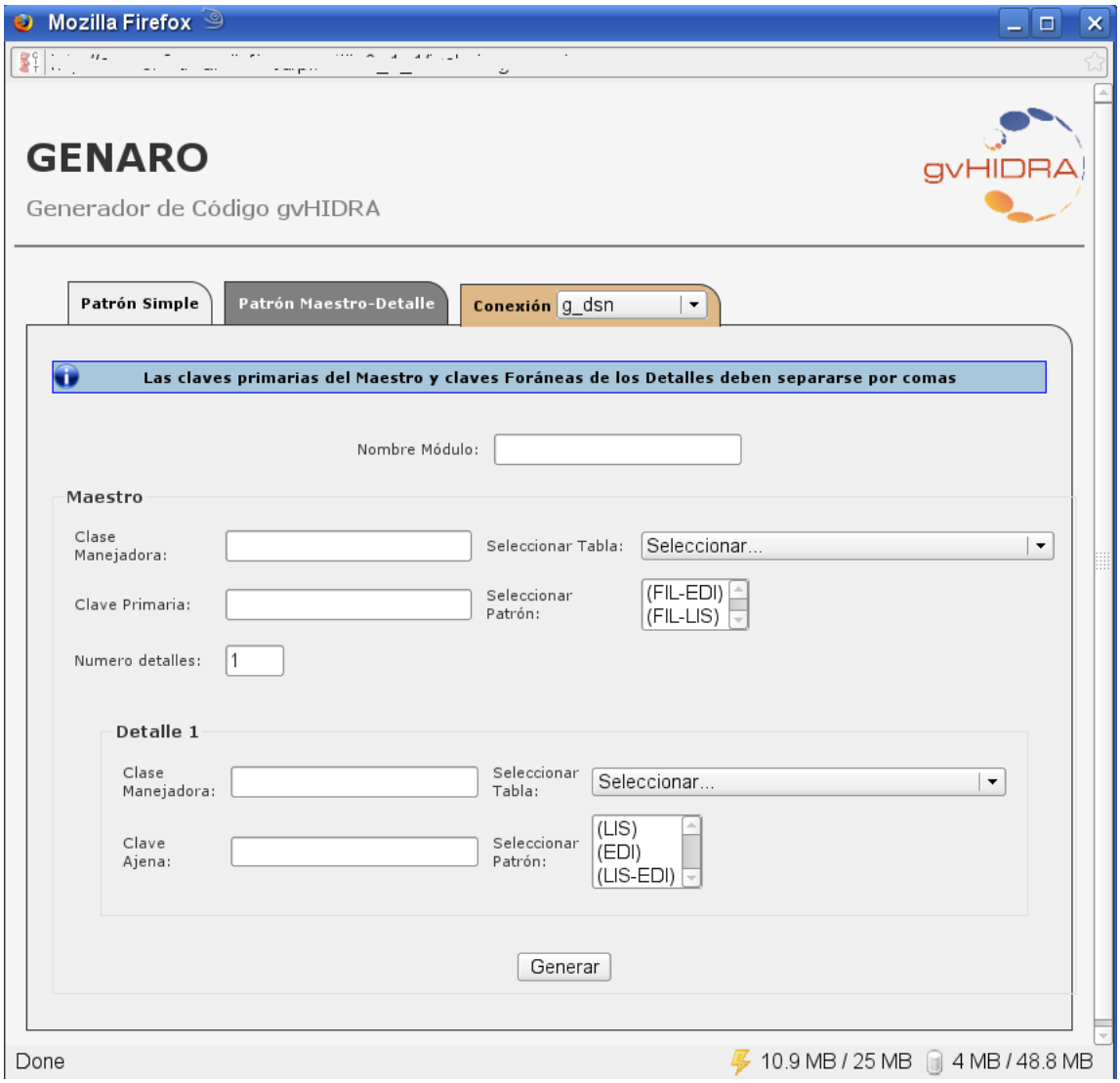
En las nuevas versiones, se puede parametrizar el mantenimiento.

Vamos a nuestra aplicación. Entramos de nuevo desde la validación (es importante porque borramos caches y cookies). Vemos que se ha creado una nueva entrada de menú y el resultado será algo como esto:

Nota

Recomendamos ver en el código fuente todos los ficheros generados para entender la estructura del código.

Para generar patrones maestro detalle tenemos la siguiente ventana



Debemos indicar los siguientes parámetros:

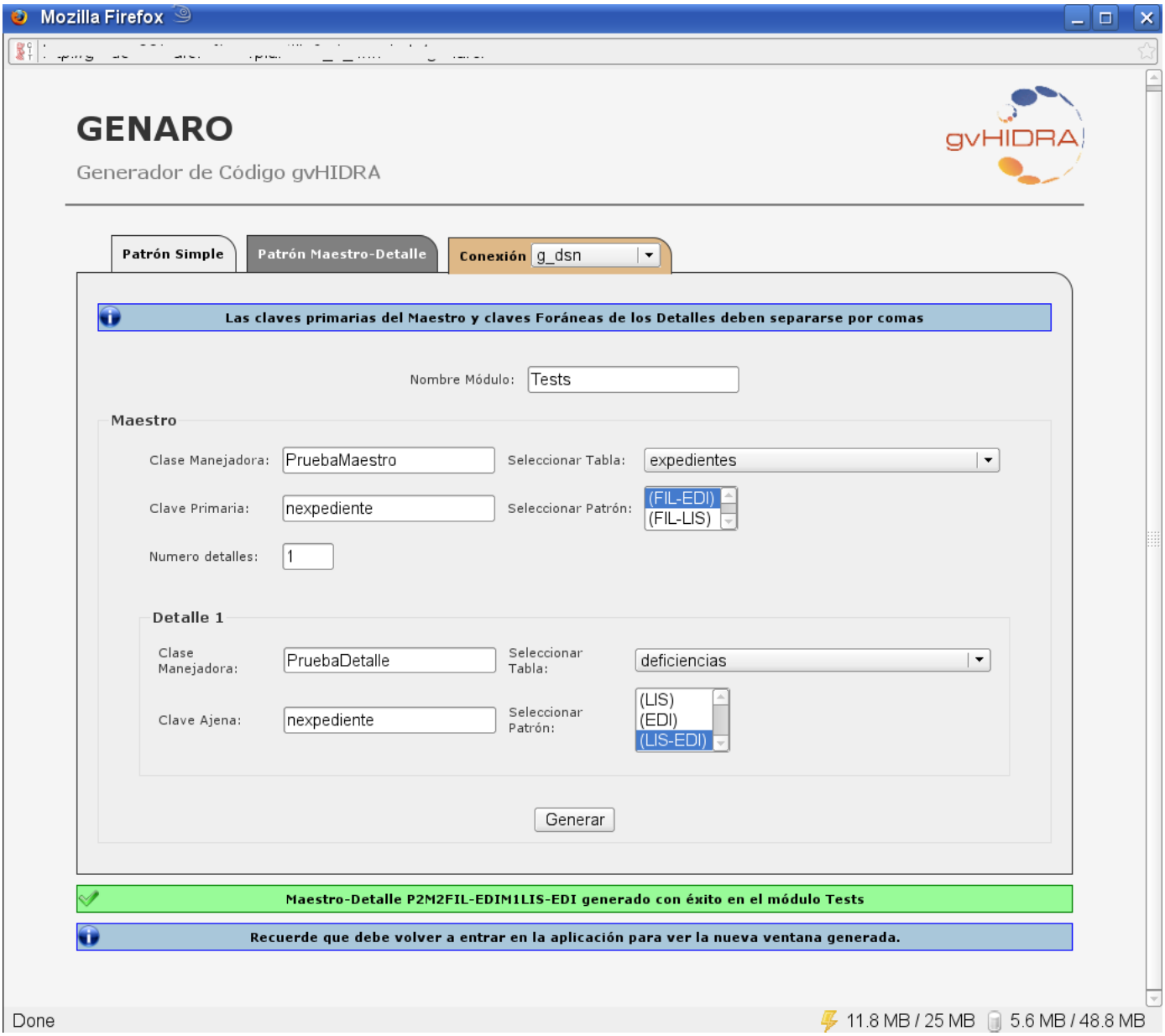
- Nombre del módulo: es un agrupador de mantenimientos. Nos permite agrupar los mantenimientos de nuestra aplicación de forma que sea más sencillo su manejo.
- Seccion Maestro:
  - Clase Manejadora: el nombre de la clase del maestro
  - Tabla: tabla de la base de datos del maestro.
  - Clave Primaria: campos que componen la clave primaria del maestro.
  - Patrón: Seleccionamos la forma de visualización del maestro (Tabular o Registro).
  - Numero de detalles: número de detalles que va a tener nuestra ventana.
- Seccion Detalle:
  - Clase Manejadora: el nombre de la clase del detalle

- Tabla: tabla de la base de datos del detalle. El sistema intentará encontrar las tablas con las que el maestro tiene relación. En caso de no encontrarlo, se puede especificar manualmente marcando el check contiguo al campo.

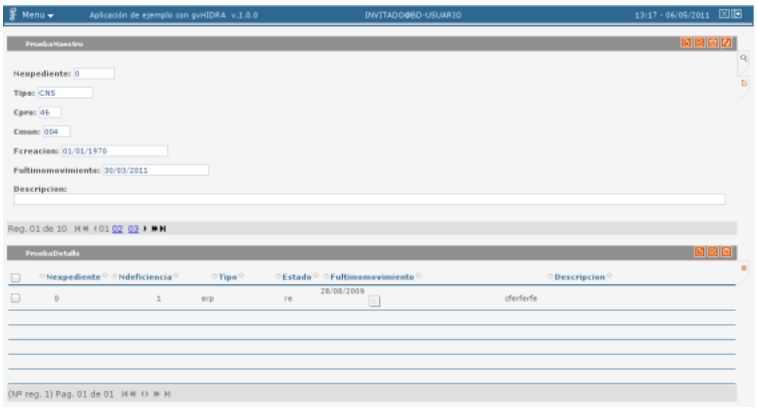
- Clave Ajena: campos que relacionan la tabla con el maestro. Corresponde con los campos que son clave ajena referenciada a los campos de la clave primaria de la tabla maestro. Si son varios, debe respetarse el orden que se ha indicado en el maestro.

- Patrón del detalle: Seleccionamos la forma de visualización del detalle (Tabular, Registro o Tabular-Registro).

Pulsamos generar y aparece la siguiente ventana



Vamos a nuestra aplicación. Entramos de nuevo desde la validación (es importante porque borramos caches y cookies). Vemos que se ha creado una nueva entrada de menú y el resultado es el siguiente:



3.2.3. Revisión de un mantenimiento

Vamos a hacer un pequeño recorrido sobre una pantalla de forma manual. Esto puede ser de mucha utilidad para todos los que estan empezando, para aprender la estructura de un proyecto gvHIDRA.

1. En primer lugar tenemos que saber con que tipo de pantalla trabajamos, con ello sabremos exactamente cuantos ficheros utilizaremos para la parte de *control de negocio* (directorio *actions*). Este fichero php (de aquí en adelante le llamaremos **clase manejadora**) será el que controlará la lógica de negocio, toda ella o puede compartirla apoyándose en clases de negocio adicionales cuando la aplicación lo requiera. Estas clases adicionales, por organización, las localizaremos en un directorio llamado *class* al nivel de actions.

   Los patrones los clasificamos en dos, simples y complejos. En los primeros entra el patrón tabular, patrón registro y el tabular-registro, y en los segundos, los maestro-detalle y el árbol. Esta diferencia se basa en que para los simples solamente es necesaria una clase manejadora, y para los complejos es necesaria más de una. Concretamente, en los maestro-detalle necesitaremos una clase manejadora para el maestro y una por cada detalle que tengamos, y para el árbol, se necesitará una por cada opción que vaya a tener el menú lateral.

   La clase manejadora heredará de **gvHidraForm_DB**, que contiene el comportamiento general de un mantenimiento común (inserciones, modificaciones y borrados) contra un SGBD relacional. Un ejemplo tipo de un patrón Tabular podría ser el siguiente:

```
<?php

/* CLASE MANEJADORA */

class TinvEstados extends gvHidraForm_DB
{

   public function __construct()
   {

      /*manejador de conexión*/
      $conf = ConfigFramework::getConfig();
      // dsnAplicacion: será el que se haya definido en
 gvHidraConfig.xml
```

```
        $dsn = $conf->getDSN('dsnAplicacion');

        //Las tablas sobre las que trabaja
        $nombreTablas= array('tinv_estados');
        parent::__construct($dsn,$nombreTablas);

        //La select que mostramos
        $this->setSelectForSearchQuery("SELECT cestado as
 \"lisCodigoEstado\",
                                                destado as
 \"lisDescEstado\" FROM tinv_estados");
        //El orden de presentación de los datos
        $this->setOrderByForSearchQuery("cestado");

        /* Añadimos los Matching - Correspondecias elemento TPL <->
 campo de BD */
        $this-
>addMatching("filCodigoEstado","cestado","tinv_estados");
        $this->addMatching("filDescEstado","destado","tinv_estados");

        $this-
>addMatching("lisCodigoEstado","cestado","tinv_estados");
        $this->addMatching("lisDescEstado","destado","tinv_estados");
    }
}
?>
```

Del contenido de la clase podemos destacar:

- La clase hereda de **gvHidraForm_DB** (línea 5). Esto se debe a que, tal y como hemos comentado, dicha clase es la que nos proporciona el comportamiento genérico de gvHidra en relación a las acciones contra una BBDD relacional. En otros casos se puede heredar directamente de **gvHidraForm**.

- Una vez en el constructor debemos indicar la fuente de datos (líneas 12-13). Primero cargamos el fichero (**gvHidraConfig.inc.xml**) de configuración del framework, del custom y el de la propia aplicación. De esa configuración que nos ofrece el fichero cargamos el dsn que hemos definido para la aplicación, en la variable *$dsn* tendremos una referencia a la fuente de datos sobre la que se quiere trabajar. Si tenemos más de una fuente de datos, ver el capítulo Fuente de datos [148].

- Debemos llamar al constructor del padre (línea 17) e instanciar una serie de propiedades que marcarán el comportamiento del panel. Se le pasa como parámetro la conexión definida anteriormente y el nombre de las tablas que va a mantener el panel ($nombreTablas en el ejemplo).

- Debemos inicializar la consulta de la base de datos mediante el método **setSelectForSearchQuery()**. Es importante que en la select los nombres de los campos correspondan con los que luego utilizaremos en la plantilla (fichero tpl) En el ejemplo se utilizan alias para asegurar esa concordancia de nombres, esta es una medida aconsejable para evitar confusiones.

  NOTA: Ver que el alias se ha definido con un prefijo "lis" porque estamos en un patrón Tabular, la consulta nos devolverá los datos que se mostrarán en el tabular. Aconsejamos el uso de esta notación, prefijo "fil" si el campo corresponde a un panel de búsqueda, "edi" si corresponde a uno de un panel registro y "lis" si corresponde a uno de un panel tabular.

***IMPORTANTE**: No se recomienda usar en la select la palabra clave "DISTINCT", ya que cuando el gestor de base de datos es Oracle, gvHidra introduce una condición en el WHERE (usando la pseudocolumna ROWNUM) para limitar el número de filas, y el resultado sería incorrecto ya que primero se aplica el ROWNUM y después el DISTINCT.*

- Opcionalmente se puede modificar el WHERE y el ORDERBY de la consulta anterior mediante los métodos correspondientes. En el ejemplo vemos que se modifica el ORDERBY (línea 23), si quisiéramos modificar el WHERE que gvHidra realiza por defecto utilizaríamos el siguiente método **setWhereForSearchQuery()**.

- El siguiente paso a realizar es dar la correspondencia entre los nombres de los campos (en la consulta) y los nombres en la plantilla (tpl). Cuando los datos son devueltos desde el panel a la capa de negocio, se necesita realizar una conversión, ya que los nombres no tienen porque coincidir (independizamos la presentación de la fuente de datos). Para realizar este proceso (que hemos denominado matching) se debe utilizar un método heredado llamado **addMatching**. En la llamada a este método le hemos de indicar 3 parámetros, que por este orden, corresponden a: nombre del campo en la TPL, nombre del campo en la consulta (el nombre del campo o el alias utilizado en la SELECT) y nombre de la tabla de la BD a la que pertenece. Para evitar problemas con los nombres de columnas entre distintos SGBD (oracle y postgresql por ejemplo), conviene usar siempre el 'as' en los nombres de columna, usando un alias preferiblemente en minúsculas.

ATENCIÓN: Todo lo expuesto se corresponde con un panel con sólo *dos modos* (tabular o registro). Si se tiene un panel con *tres modos* (tabular-registro) se deben inicializar además las siguientes propiedades:

- Utilizar el método **setPKForQueries()** para indicar que campos forman parte de la clave primaria del mantenimiento. Estos campos son los que se utilizarán para realizar el paso del modo de trabajo TABULAR al modo FICHA (registro).

- Inicializar la SELECT que contiene la consulta a realizar en el panel de FICHA con el método **setSetSelectForEditQuery()**. Es decir, la select que se lanzará cuando tras seleccionar una o más tuplas en tabular y pulsamos el botón editar, nos pasará a la FICHA.

- Opcionalmente se puede rellenar la WHERE y el ORDERBY de la edición con los métodos correspondientes, **setWhereForEditQuery()** y **setOrderByForEditQuery()**.

Hasta aquí tenemos una clase manejadora muy sencillita para una pantalla muy básica. Podemos tener la necesidad de que la interfaz sea un poco más dinámica, por ejemplo, tener listas y campos dependientes, explicado al final de este capítulo y en el capítulo 4 Elementos de pantalla avanzados [112].

También es importante que el programador sepa de la existencia de algunos métodos que puede sobrecargar para dotar de cierto comportamiento particular a las funciones genéricas de gvHidra. Por ejemplo, puede ser muy útil realizar validaciones previas a una operación en la base de datos o especificar restricciones complejas en la WHERE de una consulta dependiendo de los valores seleccionados por el usuario. Estos métodos son los explicados en el capítulo 1, punto 2 Lógica de negocio [22].

Antes de pasar al siguiente paso vamos a dar un ejemplo de estos métodos. Concretamente, vamos a forzar que cuando se inserten tuplas en nuestro ejemplo, la descripción aparezca siempre en mayúsculas.

```
public function preInsertar($objDatos)
{
   while($v_datos=$objDatos->fetchTupla())
   {
      $v_datos['descEstado'] = strtoupper($v_datos['descEstado']);
```

```
            $objDatos->setTupla($v_datos);
        }
        return 0;
    }//Fin de PreInsertar
```

2. El siguiente paso es indicar al controlador que clase es la encargada de realizar las distintas operaciones del panel y donde se debe ir dependiendo del resultado de la operación realizada. Para ello editamos el fichero **mappings.php** (directorio include de la aplicación) y añadimos las operaciones que correspondan a dicho panel.

   Para cada operación necesitamos definir quien gestiona cada operación, y donde debe redirigirse en cada posible respuesta de esa operación, si es correcta o no la respuesta.

   Con la función **_AddMapping** indicamos que clase se encarga de gestionar cada operación. De esta función los parámetros que nos interesan son los dos primeros. Con el primero indicamos el nombre de la operación, este nombre debe seguir el siguiente patrón *nombreClaseManejadora__nombreAccion*, los dos nombres están separados por dos subguiones. El segundo parámetro indica el nombre de la clase manejadora que tratará las operaciones (en el ejemplo TinvEstados).

   En *nombreAccion* nos podemos encontrar las siguientes palabras reservadas:

   - **iniciarVentana**: Acción que se lanzará cuando se quiera incluir en un panel algo que tenga que ser cargado desde BD o alguna comprobación previa a la carga de la ventana. Si se produce algún error (gvHidraError) en este momento nos redirigirá a la pantalla de entrada de la aplicación (igep/views/aplicacion.php).

     Por ejemplo poner en un panel de búsqueda una lista desplegable cargada desde BD.

   - **buscar**: Acción que normalmente se dispara desde los paneles de filtro. Comprueba si la búsqueda tiene parámetros y lanza la SELECT definida en la clase manejadora para la búsqueda.

   - **operarBD**: Acción que realiza las tres operaciones básicas de un mantenimiento: Insertar, Borrar y Modificar. Esta acción es exclusiva para trabajar con patrones de un solo modo, patrón Tabular o patrón Registro con o sin búsqueda.

   - **nuevo**: Acción que se lanza al pulsar al botón de nuevo registro en un panel. Se utiliza para cargar listas u otros componentes, inicializar datos antes de que el usuario empiece la inserción de datos.

   - **editar**: Acción que se lanza al pulsar el botón modificar de un panel, concretamente cuando nos encontramos en un Tabular-Registro. Se lanzará la consulta definida para la edición en la clase manejadora correspondiente.

   - **insertar**: Acción exclusiva para cuando se trabaja con un patrón Tabular-Registro. Se lanza cuando se va a insertar el registro nuevo.

   - **modificar**: Acción exclusiva para cuando se trabaja con un patrón Tabular-Registro. Se lanza cuando se van a guardar las modificaciones hechas en el panel.

   - **borrar**: Acción exclusiva para cuando se trabaja con un patrón Tabular-Registro. Se lanza cuando se va a efectuar la operación de borrado.

   - **cancelarTodo**: Acción que, como su propio nombre indica, cancela todo, es decir, elimina el contenido de la última consulta y de la última edición.

   - **cancelarEdicion**: En este caso, esta acción solamente elimina el contenido de la última edición.

- **recargar**: Acción exclusiva para ventanas maestro-detalle. Acción que se lanza al paginar en un maestro para recargar los detalles.

Ahora tenemos que definir las redirecciones dependientes de las respuestas a la operación, para ello tenemos el método **_AddForward**. Este método necesita de tres parámetros, siendo el primero el mismo que para la función _AddMapping, el segundo parámetro será una palabra clave que identifica la respuesta a la operación, y el tercer parámetro será la ruta donde se redirigirá la respuesta, normalmente será a un fichero de la presentación del directorio *views*.

Las *acciones genéricas*, que corresponden con las acciones enumeradas antes, tienen unos retornos fijos que dependerán del resultado de la operación, las palabras clave de respuesta para estas acciones son **gvHidraSuccess**, **gvHidraError**, **gvHidraNoData**, significando: todo correcto, ha habido error, no hay datos, respectivamente. Las *acciones particulares* tendrán tantos retornos como el programador crea conveniente. A continuación mostramos la parte del mappings que correspondería a nuestro ejemplo:

```
<?php

class ComponentesMap extends gvHidraMaps
{
        /**
         *      constructor function
         *      @return void
         */

        function ComponentesMap ()
        {
            parent::gvHidraMaps();
            ...
            $this->_AddMapping('TinvEstados__iniciarVentana',
 'TinvEstados', '', 'IgepForm', 0);
            $this->_AddForward('TinvEstados__iniciarVentana',
 'gvHidraSuccess', 'index.php?view=views/tablasMaestras/
p_estados.php');
            //
            $this->_AddForward('TinvEstados__iniciarVentana',
 'gvHidraError', 'index.php?view=igep/views/aplicacion.php');

            $this->_AddMapping('TinvEstados__buscar', 'TinvEstados',
 '', 'IgepForm', 0);
            $this->_AddForward('TinvEstados__buscar',
 'gvHidraSuccess', 'index.php?view=views/tablasMaestras/
p_estados.php&panel=listar');
            $this->_AddForward('TinvEstados__buscar',
 'gvHidraError', 'index.php?view=views/tablasMaestras/
p_estados.php&panel=buscar');
            $this->_AddForward('TinvEstados__buscar',
 'gvHidraNoData', 'index.php?view=views/tablasMaestras/
p_estados.php&panel=buscar');

            $this->_AddMapping('TinvEstados__operarBD',
 'TinvEstados', '', 'IgepForm', 0);
```

```
            $this->_AddForward('TinvEstados__operarBD',
 'gvHidraSuccess', 'index.php?view=views/tablasMaestras/
p_estados.php&panel=listar');
            $this->_AddForward('TinvEstados__operarBD',
 'gvHidraError', 'index.php?view=views/tablasMaestras/
p_estados.php&panel=listar');
            $this->_AddForward('TinvEstados__operarBD',
 'gvHidraNoData', 'index.php?view=views/tablasMaestras/
p_estados.php&panel=buscar');
                ...
        }
}
```

3. A continuación debemos crear un archivo php en el directorio **views** que controle la presentación. En él se definen las asignaciones de los datos a la plantilla (tpl). Si se trata de un comportamiento genérico, hay que instanciar la clase **IgepPantalla** (línea 3) que define el comportamiento general de una pantalla. En la línea 5 se carga la clase **IgepPanel**, que nos permite tener accesibles las funciones de acceso al panel, pasándole dos parámetros, el primero será el nombre de la *clase manejadora* que corresponde a ese panel, y el segundo, *smty_datosTabla*, es el nombre de una variable smarty que se habrá definido en la tpl correspondiente al panel, en el siguiente punto se explica. Una vez aquí tenemos que activar las pestañas laterales que nos permitirán cambiar de modo (en el ejemplo búsqueda/tabular), esto lo haremos con el método **activarModo**, al que se le pasan dos parámetros, el primero es el panel al que hará referencia ("fil" -> panel búsqueda, "lis"-> panel tabular, "edi"-> panel registro), y el segundo parámetro es nombre de un parámetro que indica el estado de la pestaña ("estado_fil"->pestaña filtro, "estado_lis"->pestaña tabular, "estado_edi"->pestaña registro), si está activa o no.

   Una vez definido todo tenemos que agregar el panel a la ventana, esto lo haremos con el método **agregarPanel()**. Y por último, ya solo nos queda mostrar la tpl en pantalla, para ello utilizamos el método **display()**, que es un método propio de smarty, siendo la variable $s una instancia de la clase Smarty_Phrame.

   A continuación mostramos un fichero de ejemplo de una ventana con un panel de dos modos:

```
<?php
//Creamos una pantalla
$comportamientoVentana = new IgepPantalla();

//Creamos un panel
$panel = new IgepPanel('TinvEstados',"smty_datosTabla");

//Activamos las pestañas de los modos que tendremos en el panel
$panel->activarModo("fil","estado_fil");
$panel->activarModo("lis","estado_lis");

//Agregamos el panel a la ventana
$comportamientoVentana->agregarPanel($panel);

//Realizamos el display
$s->display('tablasMaestras/p_estados.tpl');
?>
```

4. Ahora vamos a diseñar la plantilla (tpl) de la pantalla en el directorio plantillas de la aplicación. En la tpl diseñaremos como quedará al final la pantalla, con una combinación de plugins propios de gvHidra,

etiquetas propias de Smarty y etiquetas HTML haremos el diseño. Las plantillas tendrán todas una estructura común a partir de la cual podremos ir particularizando nuestra pantalla.

En ella creamos los componentes necesarios para nuestra pantalla, ajustando todos los parámetros de cada plugin con los nombres dados en la clase manejadora y las acciones correspondientes del mappings. Lo veremos más detallado en el punto 3 Diseño de pantalla con smarty/plugins [65].

Precauciones a tener en cuenta a la hora de diseñar una tpl:

- Los identificadores de campos pueden estar en cualquier combinación de mayúsculas y minúsculas, aunque no debe haber dos donde sólo cambie esto (es decir, case-insensitive).
- Hay que evitar poner atributos o tags innecesarios, ya que eso se traduce en una página HTML de mayor peso. Por ejemplo, si una columna es oculta, no tiene sentido indicar su tamaño, obligatoriedad, ...
- Los carácteres especiales como acentos o eñes conviene ponerlos usando entidades HTML [http://www.etsit.upm.es/%7Ealvaro/manual/manual.html#6], para evitar problemas con la codificación.
- No conviene usar el carácter '_' en las tpls. Mejor darle otro nombre mediante alias en la select.
- Cuando en la tpl exista una ficha ({CWFicha}) habrá que distribuir los campos dentro de ella. En principio si son pocos campos con un simple salto de línea (<br />) se podrían dibujar, pero cuando se complica más se creará una tabla con los parámetros que se indican en el ejemplo, y luego distribuir los campos en celdas.
- Delante del primer campo de la celda no debemos poner ningún espacio en blanco ( )

```
<tr>
    <td>
         {CWCampoTexto ...
    </td>
</tr>
```

Solamente se pondrá en el caso de que se sitúen dos campos en una misma celda; así dejaríamos un espacio entre ellos. Ejemplo:

```
<tr>
    <td>
       {CWCampoTexto ...} {CWCampoTexto ...}
    </td>
</tr>
```

Ejemplo completo de una plantilla, hay que destacar que es importante **no repetir identificadores de elementos de pantalla** (nombres de los campos) dentro de una misma TPL. En el ejemplo, los conceptos estado y descripción estado corresponden a los campos filCodEstado/lisCodEstado y filDescEstado/lisDescEstado

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=
$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
 {CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=
$smty_customTitle}
  {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
 {/CWBarra}
```

```
{CWMarcoPanel conPestanyas="true"}

<!--*********** MODO fil ******************-->
  {CWPanel id="fil" action="buscar" method="post" estado="$estado_fil"
claseManejadora="TinvEstados"}
    {CWBarraSupPanel titulo="Estados de los bienes"}
      {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar"
actuaSobre="ficha"}
    {/CWBarraSupPanel}
    {CWContenedor}
      {CWFicha}
        <br />
        {CWCampoTexto nombre="filCodEstado" size="3" editable="true"
textoAsociado="Código" dataType=$dataType_TinvEstados.filCodEstado}
        <br /><br />
        {CWCampoTexto nombre="filDescEstado" size="10" editable="true"
textoAsociado="Descripción" dataType=$dataType_TinvEstados.filCodigoEstado}
        <br /><br />
      {/CWFicha}
    {/CWContenedor}
    {CWBarraInfPanel}
      {CWBoton imagen="50" texto="Buscar" class="boton" accion="buscar" }
    {/CWBarraInfPanel}
  {/CWPanel}


<!-- ***************** MODO lis ***********************-->
  {CWPanel id="lis" tipoComprobacion="envio" action="operarBD" method="post"
estado="$estado_lis" claseManejadora="TinvEstados"}
    {CWBarraSupPanel titulo="Estados de los bienes"}
      {CWBotonTooltip imagen="01" titulo="Insertar registros"
funcion="insertar" actuaSobre="tabla"}
      {CWBotonTooltip imagen="02" titulo="Modificar registros"
funcion="modificar" actuaSobre="tabla"}
      {CWBotonTooltip imagen="03" titulo="Eliminar registros"
funcion="eliminar" actuaSobre="tabla"}
    {/CWBarraSupPanel}
    {CWContenedor}
      {CWTabla conCheck="true" conCheckTodos="true" id="Tabla1" datos=
$smty_datosTabla}
        {CWFila tipoListado="false"}
          {CWCampoTexto nombre="lisCodEstado" textoAsociado="Cód. Estado."
editable="nuevo" size="2" dataType=$dataType_TinvEstados.lisCodEstado}
          {CWCampoTexto nombre="lisDescEstado" textoAsociado="Desc. Estado."
editable="true" size="12" dataType=$dataType_TinvEstados.lisDescEstado}
        {/CWFila}
        {CWPaginador enlacesVisibles="3"}
      {/CWTabla}
    {/CWContenedor}
    {CWBarraInfPanel}
      {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
      {CWBoton imagen="42" texto="Cancelar" class="boton" accion="cancelar"
action="cancelarTodo"}
```

```
    {/CWBarraInfPanel}
  {/CWPanel}

<!-- ***************** PESTAÑAS ************************-->
  {CWContenedorPestanyas}
    {CWPestanya tipo="fil" estado=$estado_fil}
    {CWPestanya tipo="lis" estado=$estado_lis}
  {/CWContenedorPestanyas}
 {/CWMarcoPanel}
{/CWVentana}
```

5. Registramos el fichero de actions con la nueva clase en el fichero **include.php** de la aplicación que se encuentra en el directorio *include*. Podemos hacerlo con un include o usando la carga dinámica de clases explicada en el capítulo 4 punto Carga dinámica de clases [144].

6. Finalmente incluimos la ventana correspondiente en *include/menuModulos.xml* que contiene las entradas de menú.

```
<opcion titulo="Estados" descripcion="Mantenimiento de estados de bienes"
 url="phrame.php?action=TinvEstados__iniciarVentana"/>
```

Una vez cubiertos estos pasos se puede probar la ventana entrando a la aplicación. Suerte!

3.3. Menú de una aplicación

3.3.1. Funcionamiento del menú

La pantalla de inicio de la aplicación está formada por tres partes, y cada una de ellas se define, de izquierda a derecha, en los ficheros menuModulos.xml, menuHerramientas.xml y menuAdministracion.xml que están en la carpeta include de la aplicación. Las tres siguen la misma sintaxis y tienen las mismas opciones, que veremos a continuación. Ejemplo:

El plugin CWPantallaEntrada, lee los tres ficheros para crear la pantalla de inicio. El objetivo de cada fichero es:

- **menuModulos.xml**: Define la jerarquía de módulos y opciones de la aplicación, que se utilizará tanto en la pantalla principal como en el menú desplegable de las ventanas.
- **menuHerramientas.xml**: Lista de herramientas disponibles para esta aplicación.
- **menuAdministracion.xml**: Define las herramientas para la administración de la aplicación.

Cada uno de estos ficheros tiene una DTD con la que se define la estructura de ese bloque de menús en la pantalla entrada. La primera etiqueta que hay que definir es **<menu></menu>**, esta etiqueta será un bloque que englobará todo lo que definamos para ese bloque (módulos, herramientas o administración).

La aplicacion puede dividirse en módulos, etiqueta <**modulo**>, aunque normalmente sólo lo harán aquellas de tamaño grande; una aplicacion de tamaño mediano o pequeño constará de un sólo módulo. Dentro de cada módulo pueden aparecer tanto ramas, etiqueta <**rama**>, como opciones, etiqueta <**opcion**>; las ramas expresan submenús y las opciones son las hojas o elementos que se asocian con una acción o pantalla.

La etiqueta <**controlAcceso**> sirve para controlar que partes de la aplicación son visibles o no, en función del rol o de los módulos asignados al usuario (esa información se extrae de la clase IgepSession a través de la sesion). Si este nodo no aparece, cualquier usuario validado podrá acceder. Este nodo debe ser siempre el primer hijo del nodo al que quiera asociarse (NÓTESE que en el caso de opcion, debe reconvertirse la etiqueta XML y ser necesario una de apertura y otra de cierre).

A continuación se relacionan el conjunto de etiquetas utilizadas son:

La imágenes que pueden aparecer asociadas a una opción del menú se encuentran ubicadas en el directorio images/menu que debemos tener en el custom que hayamos definido para nuestra aplicación.

- **<menu>...</menu>**

  Agrupación de módulos. Sólo se pueden poner a primer nivel.

  Atributos:

  - **aplicacion**: Nombre de la aplicación.
  - **titulo**: Título que aparecerá en la cabecera del bloque correspondiente.

- **<modulo>...</modulo>**

  Agrupación de opciones y ramas. Sólo se pueden poner a primer nivel.

  Atributos:

  - **titulo**: Título de la opción, será la cadena presentada en el menu y en la pantalla principal.
  - **descripcion**: Se corresponde con la cadena que se muestra en el menú cuando detenemos el cursor encima.
  - **imagen**: (Opcional) Imagen asociada en el menú. Será una ruta relativa a un fichero gráfico en la carpeta igep/images. Si instanciamos el parámetro, deberemos comprobar que la imagen existe. Si no especificamos la imagen para el módulo, por defecto se utilizará la siguiente imagen
  - **iconCSS**: (Opcional) Icono asociado en el menú. Corresponde al selector del icono que queremos, correspondiente a uno de los selectores de las librerías que se tengan cargadas. Este parámetro prevalece sobre el parámetro "imagen". iconCSS = "glyphicon glyphicon-refresh" -> Librería Glyphicons iconCSS = "fa facheck" -> Librería Font-Awesome

- **<rama>...</rama>**

  Opción dentro de un módulo que contiene opciones o más ramas

  Atributos:

  - **titulo**: Título de la opción, será la cadena presentada en el menu y en la pantalla principal.
  - **descripcion**: Se corresponde con la cadena que se muestra en el menú cuando detenemos el cursor encima.
  - **imagen**: (Opcional) Imagen asociada en el menú. Será una ruta relativa a un fichero gráfico en la carpeta igep/images. Si instanciamos el parámetro, deberemos comprobar que la imagen existe. Si no especificamos este parámetro por defecto se utilizará la siguiente imagen

- **iconCSS**: (Opcional) Icono asociado en el menú. Corresponde al selector del icono que queremos, correspondiente a uno de los selectores de las librerías que se tengan cargadas. Este parámetro prevalece sobre el parámetro "imagen". iconCSS = "glyphicon glyphicon-refresh" -> Librería Glyphicons iconCSS = "fa facheck" -> Librería Font-Awesome

- **<opcion />**

  Opción que ya enlazará con la ventana correspondiente. NÓTESE que en el caso de que se quiera control de acceso debe reconvertirse la etiqueta XML y ser necesario una de apertura y otra de cierre (<opcion><controlAcceso>...</controlAcceso></opcion>)

  Atributos:

  - **titulo**: Título de la opción, será la cadena presentada en el menú y en la pantalla principal.

  - **descripcion**: Se corresponde con la cadena que se muestra en el menú cuando detenemos el cursor encima.

  - **imagen**: (Opcional) Imagen asociada en el menú. Será una ruta relativa a un fichero gráfico en la carpeta igep/images. Si instanciamos el parámetro, deberemos comprobar que la imagen existe. Si no especificamos la imagen para la opción por defecto se utilizará la siguiente imagen

  - **iconCSS**: (Opcional) Icono asociado en el menú. Corresponde al selector del icono que queremos, correspondiente a uno de los selectores de las librerías que se tengan cargadas. Este parámetro prevalece sobre el parámetro "imagen". iconCSS = "glyphicon glyphicon-refresh" -> Librería Glyphicons iconCSS = "fa facheck" -> Librería Font-Awesome

  - **url**: Indicaremos la clase PHP encargada de manejar la opción (se añade internamente la cadena view).

  - **ventana**: Parámetro que utilizaremos cuando queramos que la url se nos abra en una ventana diferente de la aplicación, pondremos ventana="true". Si no aparece se abrirá en la misma ventana de la aplicación.

- **<controlAcceso>...</controlAcceso>**

  Bloque para definir los roles y módulos de usuarios. Aparecerá inmediatamente después del módulo, rama u opción que haya que controlar.

- **<rolPermitido>**

  Define el rol para acceder al nodo. Se hace uso de tantas etiquetas <rolPermitido> como roles puedan acceder a ese nodo.

  Atributos:

  - **valor**: Identificador del rol

- **<moduloPermitido>**

  Módulo de acceso. No tiene nada que ver con la etiqueta módulo que se ha definido antes, y que sirve para agrupar las opciones en la aplicación.

  <moduloPermitido /> Pueden imitar el comportamiento de los roles o ser más restrictivos. Si sólo se especifica el módulo, los usuarios que lo tengan asignado podrán acceder.

<moduloPermitido>...</moduloPermitido> Si además se especifica una lista de valores, se comprobará si el usuario tiene asignado un módulo, y si lo tiene, el valor del mismo debe aparecer entre los especificados en la etiqueta <valorModulo> .

Atributos:

- **id**: Identificador del módulo

- <**valorModulo**>

  Aparecerá una etiqueta por cada valor al que se le dará acceso a la opción.

  Atributos:

  - **valor**: valor del módulo

3.3.1.1. Ejemplo con el fichero completo

```
<?xml version="1.0" encoding="iso-8859-1"?>

<menu aplicacion="nombreAplicacion">

    <modulo id="1" titulo="modulo Uno" imagen="menu.gif">
        <controlAcceso>
            <rolPermitido valor="INFORMATICO"/>
            <rolPermitido valor="SUPERNENA"/>
            <rolPermitido valor="MIEMBROIGEP"/>
            <moduloPermitido id="P_TRAMITA" />
            <moduloPermitido id="david">
                <valorModulo valor=" " />
            </moduloPermitido>
            <moduloPermitido id="DIASEMANA">
                <valorModulo valor="LUNES" />
                <valorModulo valor="MARTES" />
            </moduloPermitido>
        </controlAcceso>
        <opcion imagen="menu.gif" titulo="tOpcion1.1"
descripcion="dOpcion1.1" url="URLOpcion1-1.php" />
        <opcion imagen="menu.gif" titulo="tOpcion1.2"
descripcion="dOpcion1.1" url="URLOpcion1-1.php" />
        <rama titulo="ramaL2" id="r2">
            <opcion imagen="menu.gif" titulo="tOpcion2.1"
descripcion="dOpcion2.1" url="URLOpcion2-1.php">
            </opcion>
            <opcion imagen="menu.gif" titulo="tOpcion2.2"
descripcion="dOpcion2.2" url="URLOpcion2-1.php"/>
            <rama titulo="ramaL3" id="r3">
                <opcion imagen="menu.gif" titulo="tOpcion3.1"
descripcion="dOpcion3.1" url="URLOpcion3-1.php" />
                </opcion>
            </rama>
        </rama>
        <opcion imagen="menu.gif" titulo="tOpcion1.4"
descripcion="dOpcion1.4" url="http://www.google.es" />
```

```
    </modulo>

    <modulo id="2" titulo="modulo Segundo" imagen="menu.gif">
        <opcion imagen="menu.gif" titulo="tOpcion1.1"
  descripcion="dOpcion1.1" url="URLOpcion1-1.php" />
        <rama titulo="ramaL2" id="r2">
            <opcion imagen="menu.gif" titulo="tOpcion2.1"
  descripcion="dOpcion2.1" url="URLOpcion2-1.php">
                <controlAcceso>
                   <rolPermitido id="INFORMATICO"/>
                   <moduloPermitido id="P_TRAMITA"/>
                 </controlAcceso>
            </opcion>
            <opcion imagen="menu.gif" titulo="tOpcion2.2"
  descripcion="dOpcion2.2" url="URLOpcion2-1.php" />
            <rama titulo="ramaL3" id="r3">
                <controlAcceso>
                    <moduloPermitido id="COLORES">
                        <valorModulo valor="ROJO" />
                        <valorModulo valor="AZUL" />
                    </moduloPermitido>
                </controlAcceso>
                <opcion imagen="menu.gif" titulo="google"
  descripcion="Buscador" url="http://www.google.es"/>
            </rama>
        </rama>
    </modulo>

</menu>
```

3.3.1.2. Ejemplos con control de acceso

Ejemplo: Opción sin control de acceso.

```
<opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2"
 url="URLOpcion2-1.php" />
```

Ejemplo: Opción con control de acceso.

```
 <opcion imagen="menu.gif" titulo="tOpcion2.2"
 descripcion="dOpcion2.2" url="URLOpcion2-1.php">
    <controlAcceso>
        <moduloPermitido id="COLORES">
                <valorModulo valor="ROJO" />
                <valorModulo valor="AZUL" />
        </moduloPermitido>
     </controlAcceso>
 </opcion>
```

En dicho nodo se incluye tanto el tratamiento de los roles de usuario como el de los módulos.

- **Rol**: se hace uso de tantas etiquetas <rolPermitido> como roles puedan acceder a ese nodo, cada rol se identifica por el atributo valor.

  Ejemplo:

```
<opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2"
 url="URLOpcion2-1.php">
   <controlAcceso>
        <rolPermitido valor="DESARROLLO_GVHIDRA" />
   </controlAcceso>
</opcion>
```

- **Módulo de acceso**: Pueden imitar el comportamiento de los roles o ser más restrictivos. Si sólo se especifica el módulo, los usuarios que lo tengan asignado podrán acceder. Si además se especifica una lista de valores, se comprobará si el usuario tiene asignado un módulo, y si lo tiene, el valor del mismo debe aparecer entre los especificados.

  Ejemplo: Opción con lista de valores, el usuario debe tener asignado el módulo COLORES y deberá tener uno de los dos valores, ROJO o AZUL para que se permita el acceso.

```
<opcion imagen="menu.gif" titulo="tOpcion2.2" descripcion="dOpcion2.2"
 url="URLOpcion2-1.php">
    <controlAcceso>
        <moduloPermitido id="COLORES">
            <valorModulo valor="ROJO" />
            <valorModulo valor="AZUL" />
        </moduloPermitido>
    </controlAcceso>
</opcion>
```

3.3.2. Opciones predefinidas para el menu

Opciones que pueden usarse en los ficheros xml de configuración de menús, y que están implementadas en gvHidra.

3.3.2.1. Manual de Usuario Guía de Estilo

Abre una ventana emergente donde se explica el funcionamiento general de los distintos elementos de las aplicaciones realizadas con gvHIDRA. Se recomienda colocar en el menú de herramientas o de administración.

```
<opcion titulo="Manual Guía de Estilo" descripcion="Manual de
 introducción a Guía de Estilo"
        url="igep/doc/manualIGEP/indice/indice.html"
 imagen="menu/24.gif"
/>
```

También hay disponible una guía rápida:

```
<opcion titulo="Guía Rápida" descripcion="Guía rápida uso de aplicaciones"
         url="igep/custom/cit.gva.es/doc/guiaRapida/
guiaRapidaUsoAplicacionesCIT.pdf"
         abrirVentana='true' imagen="menu/24.gif"
 />
```

3.3.2.2. Consola de Log o Debug de la aplicación

Abre una ventana emergente para consultar la información generada por la aplicación. Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación"
        url="igep/_debugger.php" abrirVentana="true" imagen="menu/
mensajes.gif">
   <controlAcceso>
      ...
   </controlAcceso>
</opcion>
```

3.3.2.3. Proteger contraseñas

Abre una ventana emergente donde podemos obtener el hash de una cadena (usados en servidores de web services). Se recomienda colocar en el menú de herramientas, y con control de acceso para que no esté accesible a usuarios.

```
<opcion titulo="Proteger datos usando hash" descripcion="Proteger datos usando
 hash"
        url="igep/include/igep_utils/protectdata.php" abrirVentana="true"
 imagen="menu/seguridad.gif">
   <controlAcceso>
      ...
   </controlAcceso>
</opcion>
```

3.3.3. Autenticación y autorización (módulos y roles)

3.3.3.1. Introducción

Los módulos y roles es la forma usada para controlar el acceso a las distintas partes de un aplicación. No hay que confundir estos módulos con la etiqueta <*modulo*> usados a nivel de los menús para agrupar funcionalidades.

3.3.3.1.1. Módulos

Los módulos representan permisos que un usuario tiene en una aplicación determinada. Los módulos se asignan cuando se lleva a cabo una asociación entre un usuario y una aplicación, y se cargan en el inicio de la aplicación. Cada módulo tiene un código, una descripción y opcionalmente puede tener un valor. Cada usuario puede tener asignado uno o varios módulos, y para cada uno puede modificar su valor. Algunos usos habituales de módulos son el controlar si un usuario puede acceder a una opción de menú, o si puede ver/editar un campo determinado de una pantalla, o para guardar alguna información necesaria para la aplicación (departamento del usuario, año de la contabilidad, ...).

También suelen usarse para definir variables globales para todos los usuarios, aunque en este caso es recomendable asignarlos a roles (ver apartado siguiente).

Ejemplo:

- Todos los usuarios que consulten la aplicación PRESUPUESTARIO deben tener el módulo M_CONSUL_PRESUPUESTARIO, es decir, nadie que no tenga ese módulo asociado podrá acceder al apartado de listados de la aplicación

- Sólo el personal de la oficina presupuestaria tendrá acceso a la manipulación de datos, es decir el módulo M_MANT_PRESUPUESTARIO

- Sólo técnicos y el jefe de la oficina presupuestaria tendrán tendrán acceso a la entrada de menú "control de crédito". Pues serán aquellos usuarios que tengan el módulo M_MANT_PRESUPUESTARIO, con el valor TECNICO o el valor JEFE.
- Usuarios:
  - Sandra (Administrativa de otro departamento que consulta la aplicación): Tiene el módulo M_CONSUL_PRESUPUESTARIO
  - Juan (Administrativo): Tiene el módulo M_MANT_PRESUPUESTARIO, (bien sin valor, o con el valor PERFIL_ADMD)
  - Pepe (Técnico): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO
  - Pilar (Jefa de la oficina presupuestaria): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_JEFE
- Módulos:
  - Nombre: M_CONSUL_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las consultas de la aplicación de presupuestario
    - Valores posibles: <COMPLETAR>
  - Nombre: M_MANT_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las opciones de mantenimiento de la aplicación de presupuestario
    - Valores posibles: PERFIL_ADMD, PERFIL_TECNICO o PERFIL_JEFE

3.3.3.1.2. Roles

Los roles representan el papel que el usuario desempeña en una aplicación y a diferencia de los módulos, cada usuario sólo puede tener uno y no tienen valor. ¿Entonces para que queremos los roles si podemos hacer lo mismo usando módulos sin valor? Podemos ver los módulos como los permisos básicos, y los roles como agrupaciones de módulos. Realmente los roles se utilizan para facilitar la gestión de usuarios. Si sólo usamos módulos y por ejemplo tuviéramos 30 módulos, seria muy difícil clasificar a los usuarios ya que seguramente cada uno tendría una combinación distinta de módulos, y cada vez que hubiera que dar de alta un usuario tendríamos que tener un criterio claro para saber cuales de los módulos asignar.

Con roles es más simple, ya que es sencillo ver los permisos de cualquier usuario (viendo el role suele ser suficiente), y para añadir nuevos usuarios normalmente solo necesitaremos saber su role. Para que esto sea fácil de gestionar, tendríamos que tener algún mecanismo que nos permitiera asignar módulos a roles, y que los usuarios con un rol "heredaran" estos módulos. Lo más flexible seria tener esta información en tablas aunque también se podría empezar haciéndolo directamente en PHP (o ver abajo módulos dinámicos):

```
if ($role == 'admin') {
   // combinacion 1
   $modulos[] = array('borrarTodo'=>array('descrip'=>'borra todo',));
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion11'=>array('descrip'=>'opcion 11',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   $modulos[] = array('opcion13'=>array('descrip'=>'opcion 13',));
   ...
```

```
} elseif ($role == 'gestor') {
   // combinacion 2
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   ...
} elseif ($role == '...') {
...
```

De esta forma, añadir un nuevo usuario de tipo administrador consistiría simplemente en indicar su role, en vez de tener que asignarle los N módulos que tienen los administradores.

La solución más flexible seria usar sólo módulos para controlar cualquier característica que pueda ser configurable por usuario, y asignar los módulos a los roles de forma centralizada (bien en base de datos o en el inicio de la aplicación). Asi el programador solo tendría que tratar con los módulos, y el analista con los módulos de cada role.

El ejemplo anterior usando roles podría ser:

- módulos: M_CONSUL_PRESUPUESTARIO, M_MANT_PRESUPUESTARIO (ambos sin valor)
- roles:
  - PERFIL_ADMD (módulo M_MANT_PRESUPUESTARIO), asignado a Juan
  - PERFIL_TECNICO (módulo M_MANT_PRESUPUESTARIO), asignado a Pepe
  - PERFIL_JEFE (módulo M_MANT_PRESUPUESTARIO), asignado a Pilar
  - PERFIL_OTROS (módulo M_CONSUL_PRESUPUESTARIO), asignado a Sandra

En este caso las dos soluciones tienen una complejidad similar, aunque las diferencias serán más evidentes conforme aumenten el número de módulos y usuarios. Si por ejemplo decidiéramos que ahora todos los técnicos han de tener un nuevo módulo X, sin usar roles tendríamos que añadir ese módulo a todos los usuarios que tuvieran el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO; usando roles seria simplemente añadir el módulo X al role PERFIL_TECNICO.

3.3.3.2. Uso en el framework

El primer ejemplo de uso de módulos puede encontrarse en la creación de los ficheros xml que da lugar a la pantalla de inicio de la aplicación, en dichos ficheros se utiliza la etiqueta "controlAcceso" para indicar que módulos necesita tener asignados un usuario para acceder a la opción. Por ejemplo:

```
<opcion titulo="Material Sensible" descripcion="Material Extremadamente
 Sensible"
        url="phrame.php?action=MaterialSensible__iniciarVentana"
 imagen="menu/24.gif">
   <controlAcceso>
        <moduloPermitido id="M_VIP"/>
   </controlAcceso>
</opcion>
```

Con el párrafo anterior de XML, se indica que la entrada de la aplicación "Material Sensible" sólo estará disponible para aquellos usuarios que cuenten entre sus módulos el M_VIP. También se puede hacer el control usando un role.

Los módulos podemos usarlos en otros sitios, y podemos acceder a ellos a través de los métodos:

Tabla 3.1. Listado de Métodos 1

método	descripción
IgepSession::hayModulo(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo
IgepSession::dameModulo(<modulo>)	Información del módulo
ComunSession::dameModulos()	Todos los módulos (estáticos) asignados al usuario (se recomienda usar el método con el mismo nombre de IgepSession)

Cuando queremos usar módulos que no estén permanentemente asignados a un usuario, sino que la asignación dependa de otras circunstancias que puedan ir cambiando durante la ejecución de la aplicación, la asignación no la haremos en el inicio de la aplicación. Para poder añadir o eliminar módulos en tiempo de ejecución, y conseguir, entre otras cosas el poder hacer accesibles u ocultar opciones de menú dinámicamente, podemos además usar:

Tabla 3.2. Listado de Métodos 2

método	descripción
IgepSession::anyadeModuloValor(<modulo>, <valor>=null, <descripcion>=null)	Añade un nuevo módulo al usuario
IgepSession::quitaModuloValor(<modulo>, <valor>=null)	Quita el módulo al usuario; si se le pasa valor, sólo lo quita si el valor del módulo coincide con el valor pasado
IgepSession::hayModuloDinamico(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo dinámico
IgepSession::dameModuloDinamico(<modulo>)	Información del módulo dinámico
IgepSession::dameModulosDinamicos()	Todos los módulos dinámicos asignados al usuario
IgepSession::dameModulos()	Todos los módulos asignados al usuario

Estos módulos les llamaremos módulos dinámicos. A efectos del framework, no hay diferencia entre los módulos cargados inicialmente y los módulos dinámicos. El plugin **CWPantallaEntrada** ya incluye dicha funcionalidad, por lo que si en alguno de los ficheros XML aparece una opción como esta:

```
<opcion titulo="Prueba de módulo dinamico"
      descripcion="Ejemplo de activación y descativación de opciones
 en ejecución"
      url="http://www.gvpontis.gva.es" imagen="menu/24.gif">
   <controlAcceso>
      <moduloPermitido id="MD_GVA"/>
   </controlAcceso>
</opcion>
```

Hasta que en algún punto de nuestro código PHP no realicemos una llamada como esta:

```
IgepSession::anyadeModuloValor("MD_GVA")
```

la opción permanecerá oculta.

Si después de habilitarla queremos volver a ocultarla, basta con ejecutar la opción:

```
IgepSession::quitaModuloValor("MD_GVA")
```

Podemos obtener el role del usuario con el método **IgepSession::dameRol()**.

Nota

Restricciones en los menús

Para que los cambios sean efectivos de forma visual en los menús, es necesario que se reinterprete el código XML, cosa que sólo se hace cuando se pasa por la pantalla principal de la aplicación, mientras tanto NO será actualizada la visibilidad/invisibilidad de las distintas ramas, módulos y/o opciones. Por tanto, convendrá también añadir en las acciones que correspondan al menú, comprobación explícita de que se tienen los módulos/roles necesarios para el acceso.

3.4. Diseño de pantalla con Smarty/plugins

3.4.1. ¿Qué es un *template*?

Es un fichero de texto con extensión TPL. La idea original de Smarty es que fuese una plantilla o TEMPLATE (de ahí su extensión). En el caso de gvHidra, se intentaba que la Web se pareciese lo máximo posible a un modelo Cliente-Servidor, con una estructura de capas para distanciarse lo máximo posible el diseño.

El fichero .tpl, es una plantilla en la que se diseña la parte de presentación combinando algunas etiquetas HTML (opcionalmente) con etiquetas propias de Smarty y elementos que denominamos "plugins" creados para gvHidra. Smarty actúa como motor de esas plantillas, y renderiza las etiquetas y plugins en código HTML y Javascript.

Smarty está implementado en PHP, por lo que se emplea ese lenguaje para procesar la presentación, es decir, los plugins lo utilizan para la lógica de presentación de una plantilla. Esto nos permite obtener una clara separación entre la aplicación lógica y el contenido en la presentación. No hay problema entre la lógica y su plantilla bajo la condición que esta lógica sea estrictamente para presentación.

3.4.2. Cómo realizar un *template* (tpl)

Los elementos que aparecen entre llaves dentro de la tpl pueden ser tanto plugins propios de gvHidra como etiquetas propias de Smarty. Principalmente utilizaremos plugins para diseñar la presentación de la aplicación, aunque en momentos puntuales puede ser que necesitemos el uso de alguna etiqueta de smarty.

Tenemos dos tipos de plugins:

1. *Plugin block:* Son plugins que pueden anidar otros plugins. Se componen de una etiqueta de apertura y una de cierre. En la etiqueta de apertura es donde se parametrizará el plugin.

   Ej. **{CWTabla** conCheck="true" seleccionUnica="true" id="Tabla1" datos=$smty_datosTabla**}**

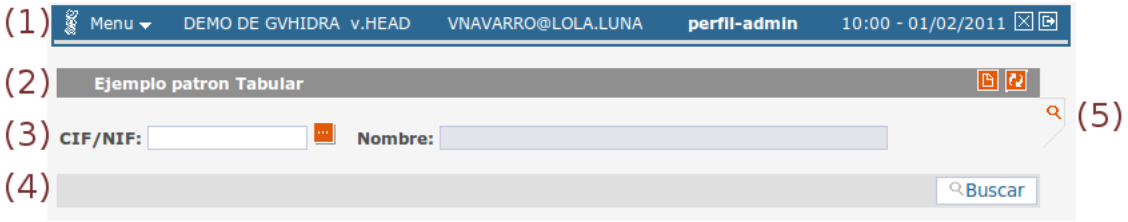
   ...

   {/CWTabla}

2. *Plugin function:* Son plugins independientes, pueden estar dentro de un plugin block. Estos solamente tienen una etiqueta, y será en ella donde parametrizaremos el plugin.

   Ej. **{CWCampoTexto** nombre="lisCoche" editable="true" size="10" textoAsociado="Coche"**}**

La tpl deberá tener una estructura válida, es decir, hay ciertos plugins que no se deben colocar de forma aleatoria, sino que tienen su orden. En el punto 1.1 del capítulo 2 Anatomia de una ventana gvHidra se explicó las zonas que existen en una pantalla gvHidra, podemos decir que en la tpl se refleja esa distribución de la pantalla.



La imagen anterior nos muestra una pantalla, ella estará englobada por el plugin **{CWVentana}** ... **{/CWVentana}**

- **{CWBarra} ... {/CWBarra}** *(1)*

  En este bloque tendremos el menú de la aplicación, si lo hay, con el plugin **{CWMenuLayer}**. A continuación hay información opcional, que podrá ser dada con los parámetros del plugin CWBarra. Los botones del final de la barra son fijos de gvHidra, la X vuelve al menú principal y el otro nos sacará de la aplicación.

  Dentro de este bloque podremos colocar botones tooltip, si son necesarios, que se corresponden con el plugin **{CWBotonTooltip}**

- **{CWMarcoPanel} ... {/CWMarcoPanel}** *(2) (3) (4) (5)*

  Este plugin engloba todo lo que consideramos panel de la pantalla. Dentro de él se diferenciarán los distintos modos de trabajo, cada modo de trabajo, *(2) (3) (4)*. irá englobado por las etiquetas **{CWPanel} ... {/CWPanel}**.

- **{CWBarraSupPanel} ... {/CWBarraSupPanel}** *(2)*

  Dentro de este bloque podremos insertar el plugin **{CWBotonTooltip}** para crear los botones de la derecha.

- **{CWContenedor} ... {/CWContenedor}** *(3)*

  Aquí es donde diseñaremos la parte principal de la pantalla, donde distribuiremos los campos, listas... donde ya utilizaremos unos plugins u otros dependiendo de si queremos diseñar un tabular o un registro.

  Tabular Necesitaremos insertar el plugin {CWTabla} al que se le pasarán ciertos parámetros, entre ellos está el más importante, *datos*, que será el que contendrá todos los datos.

  Tendrá una estructura como la siguiente:

  {CWTabla} **{CWFila}**

  ...

  {/CWFila}

  {/CWTabla}

  Registro Necesitaremos insertar el plugin {CWFichaEdicion} al que se le pasarán ciertos parámetros, entre ellos está el más importante, *datos*, que será el que contendrá todos los datos.

Tendrá una estructura como la siguiente:

{CWFichaEdicion} **{CWFicha}**

...

{/CWFicha}

{/CWFichaEdicion}

Un caso especial será cuando estamos diseñando la búsqueda, en este caso tenemos un {CWFicha} pero sin necesidad de estar dentro de un bloque {CWFichaEdicion}.

Búsqueda Tendrá una estructura como la siguiente:

{CWFicha} ...

{/CWFicha}

En esta zona es donde conoceremos la totalidad de los datos con los que se va a trabajar, por eso tenemos la excepción de que el plugin **{CWPaginador}** se ubica dentro de este bloque aunque físicamente luego se vea en el bloque del **{CWBarraInfPanel}**.

- **{CWBarraInfPanel} ... {/CWBarraInfPanel}** *(4)*

  En esta barra se incluirán, si se quieren, los botones que se corresponden con el plugin **{CWBoton}**.

- **{CWContenedorPestanyas} ... {/CWContenedorPestanyas}** *(5)*

  Este bloque engloba las pestañas que aparecerán en la pantalla, tendremos que insertar un plugin **{CW-Pestanya}** por cada pestaña que queramos.

Hay una serie de plugins que serán obligatorios y otros opcionales, que añadiremos o quitaremos según la funcionalidad que se le quiera dar a la pantalla. En la documentación de los plugins [254] se encuentra la definición de cada uno, así como la serie de argumentos que contendrán.

Nota

Tener en cuenta a la hora de diseñar la plantilla la resolución de pantalla de los usuarios. Si por ejemplo tienen 800x600 nos caben unas 12 líneas en una tabla, a diferencia de las 32 que nos caben en una resolución 1280x1024 .

Vamos a presentar la estructura obligatoria para los principales modos de trabajo:

3.4.2.1. Estructura para mantenimientos generales

```
{CWVentana ...}
   {CWBarra ...}
      si existe menú habrá {CWMenuLayer}
   {/CWBarra}

   {CWMarcoPanel ...}

      PANEL DE FILTRO
      {CWPanel id="fil" ...}
         {CWBarraSupPanel ...}
```

```
            Si existen botones ToolTip {CWBotonToolTip ...}
         {/CWBarraSupPanel}
         {CWContenedor}
            {CWFicha}

            {/CWFicha}
         {/CWContenedor}
         {CWBarraInfPanel}
            Si existen botones {CWBoton ...}
         {/CWBarraInfPanel}
      {/CWPanel}

      PANEL DE EDICIÓN
      {CWPanel id="edi|ediMaestro|ediDetalle" ... }
         {CWBarraSupPanel ...}
            Si existen botones ToolTip {CWBotonToolTip ...}
         {/CWBarraSupPanel}
      {CWContenedor}
         {CWFichaEdicion ...}
            {CWFicha}

            {/CWFicha}
         {/CWFichaEdicion}
      {/CWContenedor}
      {CWBarraInfPanel}
         Si existen botones {CWBoton ...}
      {/CWBarraInfPanel}
   {/CWPanel}

   PANEL LISTADO
   {CWPanel id="lis|lisMaestro|lisDetalle" ...}
      {CWBarraSupPanel ...}
         Si existen botones ToolTip {CWBotonToolTip ...}
      {/CWBarraSupPanel}
      {CWContenedor}
         {CWTabla ...}
            {CWFila ...}

            {/CWFila}
         {/CWTabla}
      {/CWContenedor}
      {CWBarraInfPanel}
         Si existen botones {CWBoton ...}
      {/CWBarraInfPanel}
   {/CWPanel}

   {CWContenedorPestanyas ...}
      aparecera un {CWPestanya ... } por cada pestaña que tengamos.
   {/CWContenedorPestanyas}
 {/CWMarcoPanel}
{/CWVentana}
```

3.4.2.2. Mejorando el aspecto visual

Vamos a dar unas indicaciones a la hora de la distribución y aspecto de los campos en el panel para intentar mejorar su visualización.

1. **Distribución de campos dentro de un CWFicha.**

   Para la distribución de los campos en la ficha nos tendremos que apoyar en el lenguaje HTML. Etiquetas como
 (saltos de línea),   (espacios en blanco) y <table> (tablas) nos ayudarán a distribuir los campos en el panel. Si nos decantamos por la distribución mediante tablas, utilizar el estilo asociado llamado "*text*", ver punto Personalizando el estilo [90].

2. **Tabulación entre los campos.**

   Para facilitar la navegación entre los campos de una ventana, todos los componentes disponen de un parámetro **tabindex** que permite indicar el orden en el que se quieren recorrer. Por convenio, recomendamos asignar valores de 10 en 10 entre los componentes, esto facilita la incorporación de nuevos componentes sin problemas.

```
{CWCampoTexto tabindex="10" ...}
{CWLista tabindex="20" ...}
```

   Los campos no editables no entrarán en esta navegación con el tabulador. Si, por una acción de interfaz un campo se convierte en editable, el sistema le asignará valor del parámetro "tabindex" de la tpl. Otra opción es que el programador desde una acción de interfaz cambie dinámicamente el tabindex de un componente. Para ello dispone del método **setTabIndex**.

```
$objIU->setTabIndex('field1',30);
```

3. **Resaltado de filas en un CWTabla.**

   Para poder resaltar una fila en una tabla de gvHidra tenemos que indicárselo en la construcción de la misma (típicamente en una acción buscar método postBuscar). Para ello utilizaremos el método del objDatos **setRowColor**, el cual tiene dos parámetros: la fila y el color. Actualmente, los posibles valores de color son: 'alertRow', 'noticeRow', 'errorRow' y 'suggestionRow'. También podemos poner estilos para colorear filas pares e impares con los siguientes valores: 'evenRecord', 'oddRecord'.

   Ejemplo:

```
function postBuscar($objDatos)
{
   //Obtenemos la matriz de datos
   $m_datos = $objDatos->getAllTuplas();

   foreach($m_datos as $indice =>$tupla)
   {
      //Si el tipo de la tupla es urgente lo marcamos con un color.
      if($tupla['tipo']=='URGENTE')
         $objDatos->setRowColor($m_datos[$indice],'alertRow');
      else
         $objDatos->setRowColor($m_datos[$indice],'none');
   }

   //Guardamos la matriz de datos modificada
   $objDatos->setAllTuplas($m_datos);
```

```
}
```

3.4.3. Documentación de los plugins

La documentación respecto a los plugins, descripción, parámetros y uso la podemos ver en el Apéndice B Documentación de plugins gvHIDRA [254].

3.5. Código de la lógica de negocio

En este punto entraremos en detalle del código de la lógica de negocio explicada en el punto Lógica de negocio del capítulo 2 [22].

3.5.1. Operaciones y métodos virtuales

Vamos a adentranos en el código para entender mejor las *operaciones* de gvHidra y sus *métodos virtuales*, explicadas en el punto 2 del capítulo 1 Lógica de negocio.

Las acciones buscar, insertar, editar, borrar, modificar... son acciones generales que no contemplan las reglas de negocio particulares de una aplicación. Por ejemplo, si tenemos un panel que muestra facturas, el programador puede utilizar la acción borrar para cuando un usuario lo desee pueda borrar una factura. Pero, ¿y si quiere que sólo se borren las facturas que no estén abonadas? Para ello debemos modificar el comportamiento general de la acción. Esta modificación se hará en las clases de negocio (*actions*) sobreescribiendo uno de los *métodos abstractos* (*virtuales*) vistos anteriormente con el código particular que se desee introducir.

En general, tenemos un método para antes de realizar la operación, que nos permite cancelarla (*pre-validación*) y un método para después de la operación (*post-validación*), que se puede utilizar para realizar operaciones en otras tablas.

Métodos virtuales:

1. preIniciarVentana [73]
2. preBuscar [79], postBuscar [79]
3. preInsertar [74], postInsertar [74]
4. preNuevo [76]
5. preModificar [76], postModificar [76]
6. preBorrar [77], postBorrar [77]
7. preEditar [77], postEditar [77]
8. preRecargar [78], postRecargar [78]
9. openApp [79], closeApp [79]

Todos estos métodos tienen como parámetro una instancia de la clase IgepComunicaUsuario que proporcionará al programador un acceso total a los datos que intervienen en la operación. Este acceso a los datos vendrá dado por los siguientes métodos disponibles en cualquiera de los métodos virtuales citados:

- getValue / setValue
- getSrcImg / setSrcImg
- getOldValue
- nextTupla
- currentTupla
- fetchTupla / setTupla
- getAllTuplas / setAllTuplas
- getAllOldTuplas
- getOperacion / setOperacion

Para trabajar campo por campo, en la variable $objDatos ...:

```
//Devuelve el valor del campo de la fila actual

$objDatos->getValue('nombreCampo');

//Devuelve el valor del campo antes de ser modificado
//(ojo: devuelve el valor que tiene en la BD)

$objDatos->getOldValue('nombreCampo');

//Fija el valor del campo de la fila actual

$objDatos->setValue('nombreCampo','nuevoValor');

// Devuelve el valor del src de la imagen de la fila actual
$objDatos->getSrcImg('nombreCampo');
// Fija el atributo src del campo imagen de la fila actual
$objDatos->setSrcImg('nombreCampo');

//Devuelve el registro activo sobre el origen de datos actual (cursor)

$tupla = $objDatos->currentTupla();

//Mueve el cursor a la fila siguiente

$objDatos->nextTupla();
```

Para trabajar tupla a tupla:

```
//Devuelve un vector con el contenido de la tupla actual
//y mueve el cursor interno

$tupla = $objDatos->fetchTupla();

//Fija el contenido de la fila activa con el valor deseado.
```

```
$objDatos->setTupla($tupla);
```

Para trabajar directamente con la matriz de datos:

```
//Devuelve una matriz que contiene todas las tuplas que intervienen
//en la operación

$m_datos = $objDatos->getAllTuplas();

//Fija la matriz con las tuplas que intervienen en la operación

$objDatos->setAllTuplas($m_datos);

//Devuelve la matriz de registros original correspondiente al origen de datos
//pasado como argumento (datos inserción, modificación, borrado) o el
//preestablecido.

$m_datos = $objDatos->getAllOldTuplas();
```

Para trabajar con una matriz de datos concreta (ver acceso a datos [84]):

```
//Fija la operación que será origen de los datos con los que se trabajará
//El parámetro será la operación de la cual se quiere la matriz
//  - 'visibles': Matriz de datos visibles en la pantalla.
//  - 'insertar': Matriz de datos que serán insertados en la BD
//  - 'actualizar': Matriz de datos que van a ser modificados en la BD.
//  - 'borrar': Matriz de datos que serán borrados de la BD.
//  - 'seleccionar': Matriz de datos de la/s tupla/s seleccionada/s.
//  - 'buscar': Matriz de datos que se utilizarán para la búsqueda.
//  - 'postConsultar': Matriz de datos después de una búsqueda.
//  - 'seleccionarPadre': En patrones maestro-detalle, matriz de datos que
//        nos dará la tupla seleccionada del padre.
//  - 'iniciarVentana': Matriz que contiene todos los datos del REQUEST
//  - 'external': Matriz de datos con campos no relacionados con la matriz
//        de datos.

$objDatos->setOperacion('insertar');
```

Este método, **setOperacion**, se debe utilizar en el caso de que nos encontremos en una acción particular, ya que con él fijaremos la operación sobre qué conjunto de datos queremos trabajar.

Puede darse el caso que para realizar ciertas validaciones necesitemos datos de otro panel. Para esto tenemos una clase que permite el acceso a la información de dicho panel, que se encuentra almacenada en la sesión (IgepSession).

```
// Te devuelve un campo de la tupla seleccionada.
$campo = IgepSession::dameCampoTuplaSeleccionada('clasePanel','nomCampo');

// Te devuelve la tupla seleccionada.
$tuplaSeleccionada = IgepSession::dameTuplaSeleccionada('clasePanel');

// Te devuelve el valor de una propiedad particular de la clase
$propiedad = IgepSession::dameVariable('clasePanel','propiedad');
```

Otro de los métodos que podemos sobreescribir es el encargado de cargar parámetros en las búsquedas. Concretamente es el método setSearchParameters, con él se puede indicar a la capa de negocio algunas condiciones adicionales a la where que el framework no añade por defecto.

Por otra parte, desde el método abstracto (virtual) se pueden ejecutar sentencias SQL que actuarán sobre la conexión que tenga el panel por defecto, para ello tenemos el método **consultar** (*$this->consultar($select);*) que se ejecutará con los parámetros definidos en IgepConexion.

Vamos a describir algunos ejemplos para los diferentes *métodos virtuales*.

- **Iniciar Ventana**

  Método que se ejecuta al iniciar la ventana. Por ejemplo, si tenemos una misma clase manejadora que se accede desde dos entradas de menú diferentes, agregando un parámetro en la llamada podremos saber en qué opción de menú ha entrado.

  Ejemplo Tenemos dos entradas de menú que llaman a la misma clase, una con el parámetro tipoAcceso con valor A y otro con valor B.

```
<opcion titulo="Ventana Sin Modificar" descripcion="No permitimos modificar"
url="phrame.php?action=Clase__iniciarVentana&amp;tipoAcceso=A"/>

<opcion titulo="Ventana Con Modificar" descripcion="Permitimos modificar"
url="phrame.php?action=Clase__iniciarVentana&amp;tipoAcceso=B"/>
```

Teniendo esto en cuenta, podemos comprobar qué opción de menú ha seleccionado para acceder y, por ejemplo, activar o no ciertos controles.

```
public function preIniciarVentana($objDatos) {
    $tipoAcceso = $objDatos->getValue('acceso');
    if ($tipoAcceso=='A') {
        //No puede modificar...
    }
    else{
        //Puede modificar...
    }
    return 0;
}
```

Veamos otro ejemplo donde controlamos que se visualicen unos campos o no en el panel de búsqueda, dependiendo de si el usuario es administrador o no. En primer lugar creamos un método **preIniciarVentana** en la claseManejadora:

```
public function preIniciarVentana($objDatos) {
        $admin = IgepSession::dameRol();
        //Almacenamos en una variable de instancia si es administrador o no.
        if($admin=='Administrador')
                $this->esAdministrador = 1;
        else
                $this->esAdministrador = 0;
        return 0;
}
```

Ahora en el views tenemos que asignar la variable a la tpl:

```
$admin = IgepSession::dameVariable('<claseManejadora>','esAdministrador');
```

```
$s->assign('smty_administrador',$admin);
```

Ahora en la tpl utilizaremos la lógica de smarty para poder ocultar los campos:

```
{if $smty_administrador eq 1}
        {CWCampoTexto nombre="cif" editable="true" size="5"
 textoAsociado="CIF"}
        {CWCampoTexto nombre="nombre" editable="true" size="40"
 textoAsociado="Nombre"}
{/if}
```

Con esto tendremos que el campo cif y nombre solo aparecerán si el usuario es administrador.

- **Insertar**

  Por defecto se insertan todas las tuplas que el usuario ha introducido en el panel. Con los métodos **preInsertar**, poder validar si se continua o no con la operación, y **postInsertar**, poder realizar operaciones posteriores a la inserción, controlaremos el funcionamiento particular de este proceso.

  Ejemplo En *preInsertar* se comprueba que todas las facturas que se han introducido correspondan a un proveedor válido.

```
public function preInsertar($objDatos)
{
   while($tupla = $objDatos->fetchTupla())
   {
        //Comprobamos que el cif y el orden pertenecen o a un proveedor o a
 una factura
        $str_selectCAj = "SELECT count(1) as \"cuenta\" FROM tcom_proveed
        WHERE nif = '" .$tupla['cif']."' AND orden = ".$tupla['orden'];
        $resCAj = $this->consultar($str_selectCAj);
        if($resCAj[0]['cuenta']!=1)
        {
           //Error de validación
           $this->showMessage('APL-24');
           return -1;
        }
   }
   return 0;
}
```

Otro ejemplo típico es el del cálculo de secuencias. En el siguiente ejemplo mostramos como se calcula el número de secuencia de nuevas facturas dependiendo del año.

```
public function preInsertar($objDatos)
{
    //Para que cuando insertas dar el autonumérico
    $secFacturas['anyo']=$objDatos->getValue('anyo');
    $numEntrada = $this->calcularSecuencia('tinv_facturas','nfactura',
$secFacturas);
    do
    {
        $objDatos->setValue('nfactura',$numEntrada);
        $numEntrada++;
```

```
    } while($objDatos->nextTupla());
    return 0;
}//Fin de PreInsertar
```

El *postInsertar* se ha utilizado para mandar un correo después de la inserción del registro en la tabla:

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $indice = key($m_datos);
  if ($objDatos->getValue('dgral')!='' && $objDatos->getValue('cserv')!=''))
        $nombreServDgral = $conexionOracle->consultar("select ddg as
 \"nombreDgral\",dservc as \"nombreServicio\"
                                                        from vcmn_organoso where
 cdgo='".$objDatos->getValue('dgral')."'
                                                        and cservo='".$objDatos-
>getValue('cserv')."'");

  //Obtenemos los datos de la clave informática,el número reset y el nombre
 del expediente
  if ($objDatos->getValue('codexp')!='')
  {
        $claveinf = $conexionOracle->consultar("select numero_reset as
 \"numreset\",cianyo as \"cianyo\",cidg as \"cidg\",
                                                cinum as \"cinum\",citipo as
 \"citipo\",cinumtipo as \"cinumtipo\",
                                                objeto as \"nombre_exp\"
                                                from vopu_todos_exped where
 claveseg='".$objDatos->getValue('codexp')."'");
        if(count($claveinf)<=0)
        {
                $this->showMessage('APL-18');
                return -1;
        }
  }

  //Para poder obtener los datos del usuario de la sesión
  $datosusuario=IgepSession::dameDatosUsuario();

  //Formamos el asunto del mensaje:
  $asunto=$objDatos->getValue('aplicacion')."/".$datosusuario['nombre'];
                ...

  return parent::envioCorreo($m_datos,$rol,$indice,$datosusuario,$asunto,
$texto);

}//Fin de postInsertar
```

Otro ejemplo de postInsertar puede ser que al insertar en una tabla maestro queramos que se realice una inserción en una tabla detalle dependiente (entidad débil). Tras realizar esto,

por ejemplo, podemos decidir salir de la pantalla con un actionForward de salida, indicado para este caso especial.

```
public function postInsertar($objDatos)
{
  global $g_dsn_oracle;
  $conexionOracle = new IgepConexion($g_dsn_oracle);

  $error = $conexionOracle->operar("INSERT INTO tabla2 VALUES (1,2)");
  if($error==-1)
  {
    $this->obj_errorNegocio-
>setError("APL-15","TinvLineas2.php","postInsertar");
    return -1;
  }
  else
    return $objDatos->getForward('salir');
}//Fin de postInsertar
```

- **Nuevo**

  Tenemos el método **preNuevo**, que cuando accedemos a un panel en el que vamos a introducir datos nuevos para insertar, nos va a permitir preparar algunos datos o visualización de datos, previos a la inserción, por ejemplo, cuando tenemos algún campo que tiene un valor por defecto o es calculado a partir de los valores de otros campos.

  Ejemplo Tenemos un campo secuencial que no debe ser introducido por el usuario.

```
public function preNuevo($objDatos)
{
   $secuencia = $this->calcularSecuencia('tinv_tipo_bajas','cbaja',array());
   $objDatos->setValue('codigoBaja',$secuencia);
   return 0;
}
```

- **Modificar**

  Por defecto, en gvHidra, se realiza la actualización de todas las tuplas que el usuario ha modificado en el panel. Con los métodos abstractos **preModificar** y **postModificar** podremos modificar este comportamiento.

  Ejemplo Comprobaremos en *preModificar* si la factura tiene fecha de certificación, en cuyo caso no se puede modificar, notificándose al usuario con un mensaje. (Nota: por diseño, en este caso solo se permite la modificación de una factura cada vez, $m_datos solo contiene una tupla)

```
public function preModificar($objDatos)
{
    $indice = key($m_datos);
    if ($objDatos->getValue('fcertificacion')!="")
    {
       $this->showMessage('APL-12');
       return -1;
    }
    return 0;
```

```
}
```

Otro de los ejemplos puede ser el caso de que tengamos que comprobar el estado de una variable del maestro (utilizaremos la clase IgepSession) antes de operar en el detalle. En el ejemplo comprobamos el valor de la variable certificada del maestro para saber si se pueden modificar las líneas de una factura. Si es así, forzamos a que la descripción de las líneas se almacenen en mayúsculas.

```
public function preModificar($objDatos)
{
    //Comprobamos si está la factura certificada... en este caso cancelamos la
 operacion
    $fechaCertif =
 IgepSession::dameCampoTuplaSeleccionada('TinvEntradas2','fcertificacion');
    if ($fechaCertif!="")
    {
       $this->showMessage('APL-12');
       return -1;
    }
    $m_datos = $objDatos->getAllTuplas();
    foreach($m_datos as $indice => $tupla)
    {
        $m_datos[$indice]["dlinea"] = strtoupper($tupla["dlinea"]);
    }
    $objDatos->setAllTuplas($m_datos);
    return 0;
}//Fin de preModificar
```

- **Borrar**

  Con los métodos **preBorrar** y **postBorrar** modificaremos el comportamiento de la operación borrado.

  Ejemplo Vamos a simular el comportamiento de un borrado en cascada. Es decir, si se borra una tupla de la tabla “maestra”, se borrarán todas las tuplas de la tabla “detalle”. En este caso se borran todas las líneas de una factura antes de que se borre la factura.

```
public function preBorrar($objDatos)
{
    $this->operar("DELETE FROM tinv_bienes WHERE
        anyo ='".$objDatos->getValue("anyo")."' and nfactura=".$objDatos-
>getValue("nfactura")." ");
    $errores = $this->obj_errorNegocio->hayError();
    if($errores)
    {
        $this->showMessage($this->obj_errorNegocio->getError());
        $this->obj_errorNegocio->limpiarError();
        return -1;
    }
    return 0;
}//Fin de preBorrar
```

- **Editar**

  La operación de edición se lanza cuando se pasa de modo tabular a modo ficha en un panel mediante la acción “modificar”. Para ello se dispone de dos métodos: **preEditar** y **postEditar**. El método *preEditar*

recibe como parámetro de entrada las tuplas que se han seleccionado para pasar al modo ficha. Y el método *postEditar* recibe como parámetro de entrada el resultado de la consulta realizada, de modo que se le pueden añadir tuplas o campos.

Ejemplo Vamos a comprobar que sólo se pueda modificar un expediente si su estado es “completado”.

```
public function preEditar($objDatos)
{
   while($tupla = $objDatos->fetchTupla())
   {
      if($tupla['estado']!='completado')
      {
          $this->showMessage('APL-14');
          return -1;
      }
   }//Fin de foreach
   return 0;
}
```

Otro ejemplo donde vamos a crear un campo de tipo lista para que el usuario pueda introducirlo. El campo unidadesBaja lo tiene que crear el programador e incluirlo en el resultado de la consulta para cada uno de los registros que se hayan seleccionado.

```
public function postEditar($objDatos)
{
   //Cargamos una lista para cada una de las tuplas con los bienes que puede
 dar de baja
   $m_datos = $objDatos->getAllTuplas();
   foreach($m_datos as $indice => $linea)
   {
        $listaBajas = new gvHidraList('unidadesBaja');
        $i=1;
        while($i<=$m_datos[$indice]['unidadesDisp'])
        {
                $listaBajas->addOption("$i","$i");
                $i++;
        }
        $m_datos[$indice]['unidadesBaja'] = $listaBajas->construyeLista();
   }//Fin de foreach
   $objDatos->setAllTuplas($m_datos);
   return 0;
}
```

- **Recargar**

Esta acción se realiza cuando vamos a mostrar la información de un panel que depende de otro, es decir, cuando vamos a mostrar la información de un detalle en un maestro-detalle. Se proporcionan dos métodos para parametrizar esta acción: **preRecargar** y **postRecargar**. El primero recibe como parámetro la selección del maestro, y el segundo el resultado de la consulta.

Ejemplo Vamos a mostrar como recargar un detalle que no esté compuesto de una select. En este caso, cargamos el obj_ultimaConsulta (objeto que contiene lo que se va a mostrar en pantalla) con una matriz de datos que tenemos previamente cargada en la clase. Al tratarse de

un detalle, esta matriz de datos dependerá de la selección realizada en el padre. En nuestro ejemplo el campo clave es ccentro, y la estructura de la variable líneas es una matriz del modo [ccentro][indice][campo]. De modo que al cambiar el ccentro, sólo mostraremos las líneas que correspondan a dicho centro.

```
public function postRecargar($objDatos)
{
   $ccentroSeleccionado = $objDatos->getValue('ccentro','seleccionarPadre');
   $lineas =
 IgepSession::dameVariable('InvCabCertificadosBaja','lineasCertificados');
   $objDatos->setAllTuplas($lineas[$ccentroSeleccionado]);
   return 0;
}
```

- **Buscar**

  El método **preBuscar** se dispara antes de que se realice la consulta de búsqueda (la que viene del modo búsqueda). Recibe como parámetro un objeto *IgepComunicaUsuario*, que permite al programador navegar a través de los datos que el usuario ha introducido en el modo de búsqueda. Así, y haciendo uso de métodos como *setSearchParameters*, el programador puede añadir restricciones a la consulta a realizar como EXISTS o 'fecha BETWEEN fechaInicio AND fechaFin'

  El método *postBuscar* recibe el mismo parámetro pero con el resultado de la consulta realizada, de modo que se puede modificar el contenido de dicha matriz de datos añadiendo y/o quitando registros y/o campos.

  Los métodos relativos a la búsqueda están comentados en profundidad en la sección del uso del panel de búsqueda.

- **Abrir y cerrar aplicación**

  El framework ofrece la posibilidad de ejecutar código antes de abrir y cerrar una ventana. Para ello, tendremos que hacer uso de una clase manejadora especial que se encargará de controlar el panel principal (pantalla de entrada). En esta clase, podemos sobreescribir los métodos **openApp** y **closeApp**, y modificar el comportamiento.

  Ejemplo Mostraremos como añadir un mensaje al iniciar la aplicación (método *openApp*)

```
class AppMainWindow extends CustomMainWindow
{
   public function AppMainWindow()
   {
        parent::__construct();
        //Cargamos propiedades específicas del CS
        //Cuando estamos en desarrollo registramos todos los movimientos
        $conf = ConfigFramework::getConfig();
        $conf->setLogStatus(LOG_NONE);
        $conf->setQueryMode(2);
    }

    public function openApp($objDatos)
    {
       $this->showMessage('APL-31');
       return 0;
    }
```

```
} //Fin de AppMainWindow
```

Ejemplo 2 Mostraremos como redireccionar la entrada. Este ejemplo recrear como redirigir la entrada en caso de que el usuario conectado tenga el password caducado

```
class AppMainWindow extends CustomMainWindow
{
   public function AppMainWindow()
   {
        parent::__construct();
        //Cargamos propiedades específicas del CS
        //Cuando estamos en desarrollo registramos todos los movimientos
        $conf = ConfigFramework::getConfig();
        $conf->setLogStatus(LOG_NONE);
        $conf->setQueryMode(2);
    }

    public function openApp($objDatos)
    {
       //Validez de la entrada
       $conf = ConfigFramework::getConfig();
       $g_dsn = $conf->getDSN('g_dsn');
       $con = new IgepConexion($g_dsn);
       $res = $con->consultar('SELECT caducidad FROM tcat_usuarios WHERE id =
\''.$usuario.'\'',array('DATATYPES'=>array('caducidad'=>TIPO_FECHA)));
       $ahora = new gvHidraTimestamp('now');
       if($res[0]['caducidad']<$ahora){
           $fw = $objDatos->getForward('pwdCaducado');
           return $fw;
       }
       return 0;
    }
} //Fin de AppMainWindow
```

Previamente, en el fichero mappings.php se ha definido el siguiente forward

```
$this->_AddForward('abrirAplicacion', 'pwdCaducado', 'phrame.php?
action=CambioPassword__buscar&caducidad=1');
```

3.5.2. Uso del panel de búsqueda

3.5.2.1. ¿Qué hace la acción buscar?

Buscar es la encargada de construir la consulta (SELECT) del panel utilizando los campos del panel para construir la WHERE, creando una cadena del tipo campo=valor si estos campos no son vacíos, para que esta cadena se cree es necesario que los campos del panel de búsqueda estén definidos con el método **addMatching()** en la clase de negocio correspondiente al panel. Los métodos para realizar esta búsqueda son: setSelectForSearchQuery(), setWhereForSearchQuery() y setOrderByForSearchQuery() [48].

La acción de buscar, tal y como se ha comentado en el punto anterior Operaciones y métodos virtuales [79], el programador dispone de dos métodos que puede sobrescribir para parametrizar el comportamiento de la búsqueda, **preBuscar()** y **postBuscar()**. El primero nos puede servir para modificar la SELECT antes de lanzarla (añadir constantes, añadir un filtro a la WHERE,...) o para impedir la realización de la consulta. El segundo puede servir, por ejemplo, para añadir columnas adicionales al DBResult obtenido.

3.5.2.2. ¿Qué tipos de búsqueda realiza internamente gvHidra? ¿Cómo puedo cambiarlos?

Tal y como se ha comentado en el apartado anterior, gvHidra crea una cadena que añade al WHERE de la consulta del panel con los datos que ha introducido el usuario desde el modo de búsqueda. Esta cadena se puede formar de tres formas diferentes, dependiendo del valor que se le pase al método **setQueryMode()**, teniendo en cuenta que en ninguna de las opciones se distingue por mayúsculas/minúsculas ni por acentos u otros carácteres especiales:

- *Valor 0*: Se construye la WHERE igualando los campos a los valores.

  Quedaría como: *campo = valor*.

- *Valor 1*: Se construye la WHERE utilizando el operador LIKE.

  Quedaría como: *campo LIKE %valor%*

- *Valor 2*: Se construye la WHERE igualando los campos a los valores siempre y cuando estos no contengan el carácter % o _ , en cuyo caso se utilizará el LIKE.

NOTA: gvHidra tiene marcado por defecto el **valor 2** para ejecutar las consultas.

El programador fija el tipo de consulta general para toda la aplicación mediante la propiedad **queryMode** que se encuentra en los ficheros de configuración, gvHidraConfig.xml. Si surge la necesidad de modificar este comportamiento para un panel se puede hacer uso del método **setQueryMode()** en la clase correspondiente. En las ventanas de selección también se puede configurar la búsqueda de forma similar con el método **setQueryMode()**.

3.5.2.3. ¿Cómo inicializamos un panel sin pasar por la búsqueda?

Se puede dar el caso de que el programador quiera visualizar directamente los datos sin pasar por el panel de búsqueda previamente. Esto se puede solucionar dándole en la entrada de menu de la ventana (menuModulos.xml) la url: **phrame.php?action=ClaseManejadora__buscar**.

Con esto, se realizará la busqueda de forma automática cargando los datos sin filtros extra.

```
<opcion titulo="Estados" descripcion="Mantenimiento de Estados"
 url="phrame.php?action=TinvEstados__buscar"/>
```

3.5.2.4. ¿Cómo parametrizar la consulta?

Puede ser que surja la necesidad de parametrizar la búsqueda antes de lanzarla. Con este propósito, tenemos el método **setSearchParameters()**, que se incluirá en la pre-búsqueda, para indicarle algunos valores que bien, filtren la SELECT del panel, o que queremos que nos aparezcan. Puede darse el caso de que el filtro a introducir corresponda a un campo que no tiene matching o no sea del tipo campo=valor (is null, EXISTS,...). Por ejemplo: imaginemos que tenemos un panel de búsqueda sobre datos relativos a facturas en el que hemos introducido una lista desplegable llamada abonada que tiene 3 opciones: "Si", "No" y "". Puede darse el caso que en los datos, para saber si una factura está abonada o no, tengamos que comprobar si tiene valor el campo "fechaAbono". ¿Cómo hacemos esto? Utilizando el método setSearchParameters antes de realizar la búsqueda.

```
function preBuscar($objDatos)
{
```

```
    $abonada = $objDatos->getValue('abonada')
    if($abonada!='')
    {
        if($abonada=='Si')
            $where = 'fechaAbono is not NULL';
        elseif($abonada=='No')
            $where = 'fechaAbono is NULL';
        $this->setSearchParameters($where);
    }
    return 0;
}
```

También podemos añadir condiciones sobre campos de otras tablas relacionadas. Con el método **unDiacriticCondition()** obtenemos una condición respetando el **queryMode** del formulario (aunque descartando carácteres especiales y mayúsculas):

```
function preBuscar($objDatos)
{
    $con = $this->getConnection();
    $desc = $objDatos->getValue('descrip_linea');
    if($desc!='')
    {
        $condic_lineas = $con->unDiacriticCondition('descrip_linea',$desc,
 $this->getQueryMode());
        $where = 'id_factura in (select id_factura from lineas where '.
$condic_lineas.')';
        $this->setSearchParameters($where);
    }
    return 0;
}
```

3.5.2.5. ¿Cómo limitar los registros mostrados?

gvHidra por defecto coloca un límite de 100 registros a todas la consultas realizas a partir de la acción buscar, pero esto se puede cambiar a gusto del programador para cada uno de los paneles. Puede darse el caso que un panel tenga una SELECT muy compleja y se quiera limitar el número de registros mostrados a 20. Esto se puede hacer invocando al método de negocio **setLimit()**.

```
$this->setLimit(30);
```

Para desactivar el límite, se debe pasar como parámetro el valor -1.

3.5.2.6. ¿Cómo añadir constantes a una consulta?

Siguiendo con el ejemplo anterior, puede resultar interesante cargar constantes antes de ejecutar la SELECT. Para ello tenemos el método **addConstant()** que se encarga de añadir la constante en la Select para que aparezca en el DBResult.

```
function preBuscar($objDatos)
{
  $actividad = IgepSession::dameVariable('TinvEstados','Actividad');
  $this->addConstant('Prueba',$actividad);
  return 0;
}
```

3.5.2.7. ¿Cómo conseguir que los campos que componen el filtro mantengan el valor tras pulsar buscar?

Es bastante útil para un usuario que, tras buscar, pueda mantener en el modo filtro los valores introducidos en dicho filtro. Esto les puede saber para, por ejemplo, saber por que año han filtrado. gvHidra permite que, activando un flag, se recuerden automáticamente los valores introducidos. Para activar dicho flag (por defecto viene desactivado) debe utilizar en la clase manejadora el método **keepFilterValuesAfterSearch()**.

```
public function __construct() {
  ...
  $this->keepFilterValuesAfterSearch(true);
  ...
}
```

Para hacer uso de esta utilidad, se recomienda distinguir entre los campos del filtro y el listado/edición.

3.5.2.8. ¿Cómo parametrizar el comportamiento después de la inserción?

gvHidra, por defecto, después de realizar una inserción en el modo listado (searchMode), cambia el filtro para mostrar únicamente los nuevos registros insertados, si lo que queremos es volver a realizar la búsqueda anterior se debe invocar al método **showOnlyNewRecordsAfterInsert()** con el parámetro false.

```
public function __construct()
{
    ...
    $this->showOnlyNewRecordsAfterInsert(false);
    ...
}
```

3.5.2.9. ¿Cómo cambiar los filtros que crea el framework?

En gvHidra, al realizar una de las acciones de consulta (buscar/editar), construye una WHERE y la almacena para poder refrescar tras las operaciones de modificación. Puede que se de el caso, en el que necesitemos cambiar esos filtros para poder visualizar unos registros en concreto (p.e. en un acción particular). Para estos casos, podemos acceder/cambiar dichos filtros con los siguientes métodos:

- getFilterForSearch(): Obtiene el valor del filtro que se ha construido sobre la SearchQuery tras la acción de buscar.
- setFilterForSearch(): Cambia el valor del filtro que actúa sobre SearchQuery.
- getFilterForEdit(): Obtiene el valor del filtro que se ha construido sobre la EditQuery tras la acción de editar.
- setFilterForEdit(): Cambia el valor del filtro que actúa sobre EditQuery.

```
//Cambia el filtro para SearchQuery
$this->setFilterForSearch("WHERE id=1");
```

Nota: si estamos en una acción de interfaz, será necesario recargar la consulta bien con refreshSearch() o refreshEdit().

3.5.2.10. ¿Cómo cambiar el contenido de la búsqueda desde una acción diferente a buscar?

El framework ofrece una serie de métodos para interactuar con el contenido de la búsqueda y de la edición desde fuera de la acción de búsqueda propiamente dicho. Por ejemplo, nos puede interesar cambiar el contenido de la búsqueda una vez realizada una acción concreta. Para ello, gvHIDRA proporciona los siguientes métodos:

- getResultForSearch(): Obtiene el resultado de la última consulta lanzada para desde la búsqueda. Devuelve la matriz de datos.
- setResultForSearch(): Cambia el contenido de la búsqueda con el nuevo valor introducido por parámetro. Obviamente, debe ser una matriz de datos.
- getResultForEdit(): Obtiene el resultado de la última consulta lanzada para desde la edición. Devuelve la matriz de datos.
- setResultForEdit(): Cambia el contenido de la edición con el nuevo valor introducido por parámetro. Obviamente, debe ser una matriz de datos.

```
//Vaciar el contenido de la búsqueda
$this->setResultForSearch(array());
```

3.5.2.11. En un tabular registro ¿Cómo saltar a la edición si la búsqueda devuelve un único registro?

Para el caso concreto del patrón tabular-registro, el framework ha habilitado un mecanismo para agilizar el manejo cuando la búsqueda devuelve un único registro evitando que el usuario tenga que selecionar la tupla y pulsar al botón de edición en el tabular. Este mecanismo consiste en añadir el actionForward gvHidraSuccessOne al retorno de la acción buscar apuntando al panel edición.

Para cubrir toda la cusuística, se puede habilitar o deshabilitar este comportamiento en tiempo de ejecución con el método setJumpToEditOnUniqueRecord.

3.5.3. Acciones no genéricas

Vamos a explicar ahora como podemos incorporar en una ventana acciones no genéricas.

Es bastante común que tengamos ventanas en una aplicación que tengan un comportamiento no genérico, es decir, que no podamos resolverlas mediante una acción predefinida. Los casos con los que nos podemos encontrar son:

1. Generación de listados.
2. Mostrar ventanas emergentes.
3. procesos complejos o poco comunes (enviar un correo, procesos de actualización complejos...)

Para estos casos se ha incorporado un método virtual en las clases manejadoras que cede el control de la ejecución al programador para que pueda realizar las operaciones que estime oportuno.

3.5.3.1. Acceso a datos

Antes de explicar como podemos incorporar este tipo de acciones al framework, tenemos que hacer una referencia importante al acceso a los datos. Como ya sabréis, el framework tiene una serie de métodos de extensión para cada una de las acciones que permiten al programador acceder a los datos y modificarlos.

Estos datos a los que se accede siempre van vinculados a la acción que se ejecuta. Por ejemplo para una acción insertar obtendremos los datos que se quieren utilizar para construir la INSERT.

Ahora bien, ¿qué datos vamos a utilizar en una acción particular? Esto dependerá de lo que queramos hacer. Puede que queramos recoger los datos que el usuario haya marcado para borrado para realizar un borrado; puede que queramos los datos que el usuario acaba de introducir como nuevos,...

Por todo ello, debemos indicar al framework el conjunto de datos con el que queremos trabajar, para ello debemos utilizar (y es obligatorio para este tipo de acciones) el método **setOperation()** de la instancia de IgepComunicaUsuario proporcionada. Este método admite como parámetro un string que es el nombre de la operación sobre la que queremos trabajar. Algunas de las más importantes son:

- **insertar**: Los datos que el usuario ha introducido para insertar.
- **actualizar/modificar**: Los datos que el usuario ha introducido para modificar.
- **borrar**: Los datos que el usuario ha introducido para borrar.
- **seleccionar**: Los datos que el usuario ha seleccionado.
- **visibles**: Los datos que el usuario tiene visibles en pantalla.
- **buscar**: Los datos que el usuario ha introducido en el panel de búsqueda.
- **external**: Los valores de los campos external (ver campos especiales).

Cambiando entre las operaciones podremos acceder a los datos correspondientes.

3.5.3.2. Implementación de una acción particular

A continuación iremos relatando paso a paso como añadir acciones particulares a una ventana.

En primer lugar tenemos que añadir un estímulo en la pantalla que lance dicha acción, típicamente un botón. Para ello debemos añadir en la tpl, un botón que lance la acción. Lo indicaremos con los parámetros **accion** y **action**:

```
{CWBoton imagen="50" texto="Generar" class="boton" accion="particular"
 action="nombreDeTuOperacion"}
```

Una vez creado el disparador en la pantalla, debemos indicar que clase debe gestionar la acción y que posibles destinos tendrá de respuesta. Para ello, en el fichero *mappings.php* añadimos la siguiente información:

```
$this->_AddMapping('claseManejadora__nombreDeTuOperacion', 'claseManejadora');

$this->_AddForward('claseManejadora__nombreDeTuOperacion', 'operacionOk',
       'index.php?view=views/
ubicacioFicheroViewsClaseManejadora.php&panel=buscar');
$this->_AddForward('claseManejadora__nombreDeTuOperacion', 'operacionError',
       'index.php?view=views/
ubicacioFicheroViewsClaseManejadora.php&panel=buscar');
```

Finalmente, en la clase manejadora debemos extender el método **accionesParticulares()** para poder recibir el control de la ejecución y realizar las operaciones oportunas. El código sería:

```
class claseManejadora extends gvHidraForm
{

...
```

```
    public function accionesParticulares($str_accion, $objDatos)
    {

        switch($str_accion)
        {
            case "nombreDeTuOperacion":

                // En el ejemplo usamos:
                // -comunicacion de errores mediante excepcion.
                // -recojida de datos seleccionados en pantalla.
                // -creacion de instancias de clases de negocio (podrian
ser llamadas estaticas).
                try {
                    //Recojo valores de las tuplas seleccionadas
                    $objDatos->setOperation('seleccionar');
                    $datos = $objDatos->getAllTuplas();

                    //Llamamos a la clase de negocio
                    $f = new funciones();
                    $f->miMetodo($datos);
                    $actionForward =
$objDatos->getForward('operacionOk');
                }
                catch(Exception $e)
                {
                   //Mostramos mensaje de error
                   $this->showMessage('APL-01');
                   $actionForward =
$objDatos->getForward('operacionError');
                }
                break;

            case "nombreDeTuOperacion2":
                ...
                break;
        }
        return $actionForward;
   }
```

Antes de una acción particular SIEMPRE tenemos que indicar el tipo de datos a los que queremos acceder, lo haremos con **setOperation()**. Recordemos que tenemos diferentes operaciones (insertar, modificar, borrar, seleccionar, visibles, external...)

El método **accionesParticulares()** debe devolver un objeto **actionForward** válido. Tenemos acceso a los forwards de nuestra acción a través del método **getForward** del objeto de datos (en el ejemplo $objDatos).

Opcionalmente, existe la posibilidad de crear unos Forwards especiales propios de gvHIDRA.

- **gvHidraNoAction**Este retorno impedirá que se recargue la ventana. Este tipo de retornos es muy útil cuando estamos validando la entrada del usuario y detectamos un error. Con ellos, podemos impedir una recarga de ventana y la consecuente perdida de información.
- **gvHidraReload**Este retorno es el caso contrario del anterior. Equivale a pulsar F5 en la pantalla del navegador

Para utilizar estos retornos, se debe utilizar un actionForward de la siguiente manera:

```
$actionForward = new ActionForward('gvHidraNoAction');
```

3.5.3.3. Ejemplo de listado

Un caso típico, es el de los listados. El procedimiento es el mismo al explicado anteriormente, pero debemos añadir a la generación de listado la posibilidad de generarse en una ventana emergente. Para ello debemos hacer uso del método **openWindow** pasándole el forward de destino. Por ejemplo:

```
class claseManejadora extends gvHidraForm
{
  ...

   public function accionesParticulares($str_accion, $objDatos)
   {
        ...
        //Si todo está Ok
        //Abrimos una nueva ventana donde mostraremos el listado
        $actionForward = $objDatos->getForward('mostrarListado');
        $this->openWindow($actionForward);

        //En la ventana actual mostramos un mensaje indicando que el listado
 se ha generado
        //y volvemos a la ruta deseada
        $this->showMessage('APL-29');
        $actionForward = $objDatos->getForward('operacionOk');
    }
}
```

Para obtener mas informacion sobre los informes y Jasper Report, acudid a la documentacion en el capitulo x

3.5.4. Acciones de interfaz

3.5.4.1. Definición

Las acciones de interfaz en gvHidra es todo aquel estímulo de pantalla que se resuelve con un cambio de la interfaz sin recarga de la ventana. Es decir, típicamente el uso de tecnología AJAX para actualizar la interfaz sin que el usuario perciba una recarga de la página. Algunos ejemplos de acciones que podemos necesitar son:

- Cuando pierda el foco un campo, actualizar un campo descripción (típica descripción de claves ajenas)
- Dependiendo del valor de un desplegable (HTML select), mostrar/ocultar un campo en pantalla.
- Cuando marcamos/desmarcamos un checkbox, habilitar/deshabilitar un campo.
- ...

Para ello, el framework intenta simplificar el uso de estas acciones de forma que el programador centrará sus esfuerzos en la clase manejadora (en PHP); dejando que la herramienta se encargue de la generación de Javascript. Actualmente el framework utiliza un iframe oculto que recoge el estímulo de pantalla, lo direcciona a la clase manejadora correspondiente y recibe la respuesta en formato Javascript.

3.5.4.2. Uso de las acciones de interfaz

El primer paso para definir una acción de interfaz es indicar en la TPL que el campo tiene que disparar dicha acción.

En la tpl hemos incluido el siguiente código:

```
{CWCampoTexto nombre="cif" editable="true" size="13" textoAsociado="Cif"}
 
{CWCampoTexto nombre="orden" editable="true" size="2"
 textoAsociado="Orden" actualizaA="nombre"}
 {CWCampoTexto nombre="nombre" editable="false" size="40"}
```

Se ha añadido la propiedad **actualizaA.** Esta propiedad le indica a la clase que cuando pierda el foco el campo *orden* se tiene que lanzar una acción de interfaz, el valor de esta propiedad será el nombre del campo o campos destino de la acción de interfaz (p.ej. actualizaA="nombre, apellidos").

Ahora, en la clase, debemos indicar que *acción de interfaz* corresponderá con la pérdida de foco de dicho campo. Para ello incluimos en el constructor de la clase manejadora el siguiente código:

```
//Para las validaciones cuando introduzca el proveedor
$this->addTriggerEvent('orden','manejadorProveedor');
```

En el método se introduce el nombre del campo de la tpl que lanzará la validación y el nombre del método que el programador va a implementar para realizar la validación.

La implementación del método se hace en la clase asociada al panel:

```
public function manejadorProveedor ($objDatos) //Manejador del evento
{
  $cif = $objDatos->getValue('cif');
  $orden = $objDatos->getValue('orden');

  if($cif!='' and $orden!='')
  {
     if (!$this->validaCif($cif))//Validación léxica
     {
        $this->showMessage('APL-04',array($cif));
        return -1
     }

     $proveedor = $this->obtenerProveedor($cif, $orden);
     if ($proveedor == null) // Validación semántica
     {
        $this->showMessage('APL-24',array($cif,$orden));
        return -1
     }
     else
     {
        $objDatos->setValue('nombre',$proveedor);
     }
  }
  else
     $objDatos->setValue('nombre','');
  return 0;
```

```
}

public function validarCif ($cif, $orden)
{
...
}

public function obtenerProveedor($cif, $orden)
{
  $res = $this->consultar("select * from tcom_proveed where nif = '".$cif."'
 and orden = ".$orden );
  if(count($res)>0)
  {
     return($res[0]['nombre']);
  }
  else
  {
     return(null);
  }
}
```

En resumen, el ejemplo anterior realizará una comprobación de los campos cif y orden, una vez rellenados por el usuario (concretamente cuando pierda el foco el campo orden), lanzando el método *manejadorProveedor()* y tendrá el resultado si el cif y el orden es correcto. En caso contrario, el resultado será alguno de los mensajes de error que el programador haya capturado.

En general, la interfaz y el uso de estos métodos es parecida a la de cualquier método pre o post operación. Si devuelven 0 se considera que todo ha ido correctamente y si devuelve -1 que se ha encontrado un error.

Sin embargo, estos métodos tienen una peculiaridad especial, además de las funcionalidades de las que dispone un método pre o post operación (como sabemos, se permite el acceso a los datos que hay en pantalla mediante la variable $objDatos, se pueden realizar consultas sobre la fuente de datos del panel, ...) pueden modificar aspectos de estado de la interfaz como el contenido, la visibilidad o la accesibilidad de los componentes de pantalla. Para ello debemos hacer uso de los siguientes métodos:

- **getValue($campo)**: Obtiene el valor de un campo
- **setValue($campo,$valor)**: Cambia el valor de un campo
- **setVisible($campo,$valor)**: Cambia la visibilidad de un campo, también se puede aplicar este método a un elemento CWLabel [274].
- **setSelected($campo,$valor)**: Cambia el valor seleccionado de una lista.
- **setFocus($campo)**: Fija el foco en el elemento.
- **setEnable($campo,$valor)**: Cambia la accesibilidad de un campo, también se puede aplicar este método a un elemento CWLabel [274] cuando es de tipo enlace.
- **setIcon($campo,$valor)**: Cambiará el icono css del campo. Método que solamente actúa sobre componentes del tipo CWLabel [274], en el caso de que el label solamente sea un icono css, o una etiqueta de texto acompañada de un icono css..
- **setLabelClass($campo,$valor)**: Cambia la css asociada al componente CWLabel [274] en el caso de que sea una etiqueta.
- **posicionarEnFicha($indiceFila)**: Nos permite cambiar la ficha activa.

- **getActiveMode()**: Devuelve el modo que dispara la acción.
- **getTriggerField()**: Devuelve el campo que lanza la acción de interfaz.
- **setBttlState($idPanel, $nameBttl, $on)**: Habilita o deshabilita el botonTooltip básico ('insertar', 'modificar', 'restaurar') del panel indicado.

Ejemplo:

```
//Fijar la visibilidad de un campo
$objDatos->setVisible('nfactura',true);
$objDatos->setVisible('nfactura',false);
//Fija la accesibilidad de un campo
$objDatos->setEnable('nfactura',true);
$objDatos->setEnable('nfactura',false);
//Fija el contenido de un campo
$objDatos->getValue('nfactura');
$objDatos->setValue('nfactura','20');

//Deshabilita el boton Tooltip de inserción:
$objDatos->setBttlState('edi', 'insertar', false);

//Obtienen información sobre el modo que dispara la acción (FIL-LIS-EDI)
$objDatos->getModoActivo();
```

3.6. Personalizando el estilo

El framework suministra un método para personalizar su aspecto y funcionamiento, que en adelante llamaremos 'temas'. Los temas están ubicados en la carpeta en **igep/custom**, podemos tener varios temas y activar uno en concreto mediante el atributo **customDirName** del fichero gvHidraConfig.inc.xml del framework o de la aplicación, o con el método **ConfigFramework->setCustomDirName** (ver configuración de gvHidraConfig.xml [31]).

Con gvHidra vienen estos temas configurados:

- **blueStyle**: un aspecto más actualizado en tonos azules.
- **greyStyle**: el mismo aspecto que blueStyle pero en tonos verdes.

Vamos a pasar a explicar la estructura inicial que debe contener un tema.

3.6.1. CSS

Tendremos un directorio CSS donde tendremos la hoja de estilo a utilizar en nuestra aplicación (*apl_ALL.css*), esta es la que podremos particularizar para nuestra aplicación. También nos encontramos con las siguientes hojas de estilo:

- alertify.core.css y alertify.default.css: Utilizadas para los mensajes de aviso
- aplication-login.css: Hoja de estilos correspondiente a la pantalla de validación
- breadcrumbs.css: Hoja de estilos correspondiente a las "migas de pan" que aparecen en la pantalla de entrada si se ha configurado para que existan.
- bumpbox.css: Hoja de estilos correspondiente para imágenes que utilicen el bumpbox.
- debugger.css: Hoja de estilos correspondiente al debugger.

- font-awesome.css: Hoja correspondiente a la librería de iconos Font-Awesome
- layersmenu-cit.css: Hoja correspondiente al menú desplegable de la barra superior

3.6.2. Imágenes

En el directorio imágenes tenemos una estructura de carpetas que hay que respetar, a la vez que también hay que respetar los nombres de los fichero, ya que el framework buscará en esa ruta y con ese nombre.

Explicación de la estructura de directorios:

- **acciones**: Imágenes para los CWBoton y las flechas de ordenación en un CWTabla (14.gif y 16.gif).
- **arbol**: Imágenes que se utilizan cuando trabajemos con el patrón árbol.
- **avisos**: Imágenes que se utilizan para los diferentes mensajes que utiliza el framework (avisos, sugerencias, errores...)
- **botones**: Imágenes de los botones tooltip. Existen dos imágenes por cada botón, uno con la imagen en color para cuando el botón está activo y otra para el estado inactivo, estos llevan el sufijo "off".
- **listados**: Logo para listados.
- **logos**: Imágenes para los logos de la pantalla principal y menú desplegable de la barra superior.

  "logo.gif": Logo que aparece al final de la pantalla de entrada.

  "logoMenu.gif": Logo del menú de la barra superior de las pantallas.

  "logoSuperior.gif": Logo de fondo en la pantalla de entrada.
- **menu**: Imágenes para los menús de la pantalla principal.
- **paginacion**: Imágenes para los botones de la paginación.
- **pestanyas**: Imágenes para las pestañas de los modos de trabajo.
- **tablas**: Imágenes que se utilizan en tablas. Actualmente tenemos la imagen que nos servirá para marcar la línea de separación de la cabecera en las tablas.

3.6.3. Ficheros de configuración del custom

Tenemos cuatro ficheros que nos permitirán una configuración del custom que sobreescribirá la propia del framework, pero, que a la vez estará por debajo de la configuración de cada aplicación:

- **include.php**: fichero donde se deben incluir las clases propias del custom. Pueden ser clasesManejadoras (directorio actions), tipos (directorio types), clases de negocio (directorio class), ...
- **CustomMapping.php**: define los mappings propios del custom.
- **gvHidraConfig.inc.xml**: define la configuración estática del custom.
- **CustomMainWindow**: define la configuración dinámica del custom.

3.7. Tratamiento de tipos de datos

En un formulario podemos tener campos o columnas de varios tipos de componentes, entre otros campos de texto, combos, checkbox, etc que nos permiten ayudar al usurio a la hora de introducir información. Esto

ayuda, pero no evita que en los campos de texto tengamos el riesgo de introducir valores inadecuados. Para estos casos, el framework permite definir un tipo y algunas características más de un campo facilitando algunas tareas básicas asociadas a estos (validaciones, máscaras). Para asociar el tipo, debemos hacer uso del método **addFieldType** en la clase manejadora indicando el nombre del campo y la instancia del tipo a asociar.

Una vez que hemos definido el tipo, tenemos que enlazarlo con el campo en la template con el parámetro **dataType**.

Ejemplo:

```
# clase manejadora
$this->addFieldType('filTelefono', new gvHidraString(false, 15));

# template
{CWCampoTexto nombre="filTelefono" size="15" editable="true"
 textoAsociado="Teléfono"
              dataType=$dataType_nombreClaseManejadora.filTelefono}
```

En la clase manejadora también disponemos de un método para obtener la instancia del tipo asociada a un campo (**getFieldType**)

3.7.1. Características generales

El framework proporciona diferentes tipos de datos que permiten controlar varios aspectos de interfaz y comportamientos. Concretamente, al indicar el tipo de datos de un campo el framework nos ofrece:

- Máscaras de entrada según tipo de datos en el cliente.
- Validaciones en el servidor antes de operaciones de modificación de los mismos en la BD (modificación e inserción).
- Limitar la longitud máxima del campo en pantalla, comprobaciones a nivel cliente (maxlength) y servidor
- Comprobación de obligatoriedad a nivel de cliente y servidor.
- Ordenación en tabla por tipo de datos (sino se indica ordena siempre como cadena).
- Mostrar en pantalla características especiales según tipo de datos.

Los tipos de datos que proporciona por defecto gvHidra son:

1. **gvHidraString**
2. **gvHidraDate**
3. **gvHidraDateTime**
4. **gvHidraInteger**
5. **gvHidraFloat**

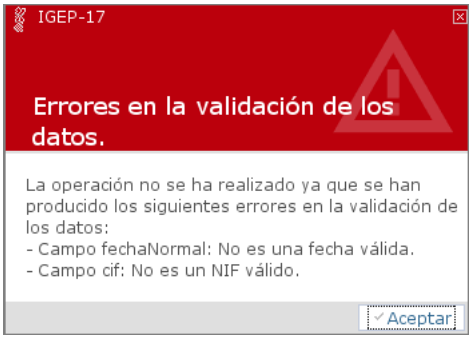
Para hacer uso de los tipos, en la clase manejadora hay que crear una instancia con el tipo que queramos, y luego asignarla al campo con el método **addTypeField()** de la clase manejadora. Todos los tipos de datos tienen unas características básicas que se pueden fijar desde el constructor con dos parámetros

1. **required:** indica que el campo es obligatorio. Los objetos pueden invocar el método setRequired para cambiar este valor.

2. **maxLength:** indica la longitud máxima esperada. Las fechas no tienen este parámetro. Los objetos pueden invocar el método setMaxLength para cambiar este valor.

Para informar al plugin (CWCampoTexto, CWAreaTexto, CWLista...) el tipo de datos que va a contener, tenemos que utilizar la propiedad **dataType**. El framework insertará los datos en la variable *$dataType_claseManejadora.nombreCampo*.

Con esta definición de tipos conseguimos que el framework valide los datos antes de cada operación de modificación de la BD (actualización e inserción). Al detectar algún error, de forma automática, recogerá la información y la mostrará en pantalla informando al usuario.



Desde la versión 4.0.0, se puede parametrizar el nombre del campo en estos mensajes. Para ello se debe hacer uso del método setLabel con el nombre de la etiqueta del campo validado.

3.7.2. Cadenas (gvHidraString)

Para el tratamiento de cadenas tenemos el tipo gvHidraString. Este tipo tiene los siguientes métodos propios:

- **setRegExp(string)**: Permite introducir una expresión regular para validar la cadena en el cliente (cuando el campo pierde el foco) como en el servidor.
- **setPasswordType(bool)**: Permite crear un campo de tipo password, de forma que oculta el contenido del mismo a los ojos del usuario.
- **setInputMask(string)**: Permite introducir una máscara de entrada de datos para la cadena. El formato de máscara será
  - '**9**' Indica un carácter numérico.
  - '**a**' Indica una letra (mayúscula o minúsculas)
  - '*****' Indica 0 o más caracteres alfanuméricos
  - '**+**' Indica, al menos, un carácter alfanumérico
  - '**(**', '**)**', '**-**', '**.**', etc: Caracteres delimitadores típicos
  - '**{n}**' Indica n repeticiones
  - '**{n,m}**' Desde n hasta m repeticiones
  - '**|**' Sentencia OR

Existen unas máscaras predefinidas, que son las siguientes:

- *Máscara*: email
  - *Placeholder*: _@_
  - *Resultado*: mimail@gmail.com
- *Máscara*: ip
  - *Placeholder*: ___.___.___.___
  - *Resultado*: 66.249.78.133
- *Máscara*: url
  - *Placeholder*: _ttp_//_
  - *Resultado*: http://www.gvhidra.org

Vamos a ver algunos ejemplos para que quede más claro el uso de las máscaras:

- *Máscara*: (99) 9999999
  - *Cadena*: 963333333
  - *Resultado*: (96) 3333333
- *Máscara*: mail: ^[a-zA-Z0-9._%-]+@[A-Z]+\\.[a-zA-Z]{2,4}
  - *Cadena*: mimail@gmail.com
  - *Resultado*: mail: mimail@gmail.com

Nota: si no se hace uso de los métodos los mismos están deshabilitados.

En el siguiente fragmento de código vemos como podemos asignar el tipo gvHidraString a un campo.

```
$telefono = new gvHidraString(false, 15);
$telefono->setInputMask('(+99)-999999999');
$this->addFieldType('filTelefono',$telefono);
```

Primero se crea un objeto de tipo string llamando a la clase gvHidraString, con el primer parámetro indicamos la obligatoriedad del campo, y con el segundo el tamaño máximo del campo. Ahora ya podemos hacer uso del método **setInputMask()**, que marcará el campo con esa máscara. Y por último se asigna el tipo al campo en concreto, en nuestro caso es "filTelefono" (alias del campo en la select de búsqueda o edición).

3.7.3. Fechas

3.7.3.1. Entendiendo las fechas en gvHidra

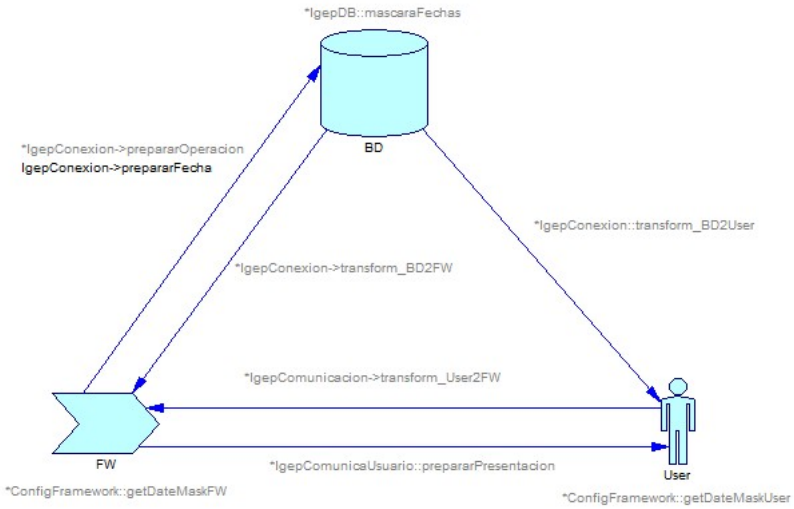
La intención de gvHidra es simplificar lo máximo posible los problemas derivados de los formatos de las fechas, y para ello se ofrece al programador una forma única de manejar las fechas (formato de negocio) y es el framework el que se encarga, en caso necesario, de convertir estas fechas al formato del usuario para interaccionar con éste (formato de usuario), o al formato que entiende nuestro gestor de base de datos para almacenar/recuperar los datos (formato de datos).

Actualmente gvHidra define de manera global el formato con el que va a mostrar las fechas al usuario (método **ConfigFramework::getDateMaskUser**) y se puede cambiar por el programador.

También se define, por cada tipo de SGBD, cual es el formato que reconocen (método **mascaraFechas** de **IgepDBMS_***) aunque no se recomienda modificarlos.

Por último, también se define el modo de representar las fechas internamente (método **ConfigFramework::getDateMaskFW**), que es el que nos permitirá operar en PHP con las fechas y que tampoco se recomienda modificarlo.

En la siguiente figura podemos ver un esquema de la situación general:



Cada nodo representa uno de los tipos de formato explicados antes, y las etiquetas junto a ellos representan el método usado para obtener la definición de ese formato. Las transiciones entre nodos representan las conversiones que se realizan en el framework, y las etiquetas junto a ellas representan los métodos que convierten del formato origen al destino de la transición. Además los métodos marcados con * (y con color más claro) indican que el método es interno al framework, y por tanto el programador no necesitará usarlo normalmente. A pesar de ello se han incluido todos ellos en el esquema para dar una visión completa.

3.7.3.2. Uso en el framework

Primero, para incluir una fecha en un panel necesitamos:

- Definir el campo en la select (búsqueda o edición).
- Definir el campo equivalente en la plantilla (tpl).
- Asociar un tipo de datos al campo de la plantilla (tpl) a través del constructor de la clase manejadora. Usaremos las clases gvHidraDate o gvHidraDatetime dependiendo de si queremos la hora o no.

```
// Fecha con calendario y etiqueta para el día de la semana en formato corto
// El parámetro indica "obligatoriedad" para cumplimentar el campo
$tipoFecha = new gvHidraDate(true);
// Existirá un calendario asociado al campo
$tipoFecha->setCalendar(true);
// Aparecerá la inicial del día de la semana (L, M, X, J, V, S, D)
$tipoFecha->setDayOfWeek('short');
$this->addFieldType('fcertificacion', $tipoFecha);
```

Cuando el programador accede a un campo declarado como fecha (en un método "pre" o "post", en una operación o en una acción particular) obtiene un objeto de la clase gvHidraTimestamp (basada en DateTime [http://php.net/manual/es/book.datetime.php]) (si no tiene valor se recibirá un NULL). No confundir esta clase con las usadas para definir el tipo, ésta es la que se usa para operar con las fechas (con o sin hora).

```
$fpeticionIni = $objDatos->getValue('fpeticionIni');
if (empty($fpeticionIni))
   return;
// en $fpeticionIni tenemos una instancia de gvHidraTimestamp

// incrementar la fecha en 1 dia
$fpeticionIni->addDays(1);

// inicializar un campo a la fecha actual:
$objDatos->setValue('fcreacion', new gvHidraTimestamp());
```

La clase gvHidraTimestamp nos ofrece algunos métodos útiles para modificar u operar sobre las fechas:

- **__construct([$ts='now' [, $tz=null]])**: el constructor tiene los mismos parámetros que DateTime; si no le pasamos ninguno se inicializa a la fecha y hora actual.
- **setTime($hours, $minutes [, $seconds = 0])**: para modificar la hora.
- **setDate($year, $month, $day)**: para modificar la fecha.
- **modify($str)**: método de DateTime que usa la sintaxis de strtotime para modificar la fecha.
- **addDays(int), subDays(int)**: añadir y restar dias a una fecha.
- **addWeeks(int), subWeeks(int)**: añadir y restar semanas a una fecha.
- **addMonths(int), subMonths(int), addYears(int), subYears(int)**: añadir y restar meses/años a una fecha. Estos métodos cambia el funcionamiento por defecto usado en modify: si estamos en dia 31 y le sumamos un mes, si no existe el dia obtenemos el 1 del mes siguiente. Con estos métodos, en la situación anterior obtenemos el último día del mes siguiente.

También hay métodos para obtener la fecha en distintos formatos o hacer comparaciones:

- **formatUser()**: obtiene la fecha en el formato que ve el usuario. En principio no debería usarse por el programador.
- **formatFW()**: obtiene la fecha en el formato que reconoce el framework. En principio no debería usarse por el programador.
- **formatSOAP()**: obtiene la fecha en el formato usado en SOAP. Lo usaremos para generar fechas en servidores de web services.
- **format($format)**: método de DateTime que acepta los formatos usados por date(). Este método sólo deberia usarse para obtener elementos de la fecha como el dia o el mes. Para otros formatos más complejos habría que valorar si se crea un nuevo método en la clase.
- **isLeap()**: indica si el año de la fecha es bisiesto.
- **cmp($f1, $f2)**: método estático para comparar fechas. Devuelve 1 si la primera en menor, -1 si es mayor y 0 si son iguales. (OBSOLETO, ya que se puede comparar los objetos directamente).
- **between($f1, $f2)**: comprueba si la fecha del objeto se encuentra entre el intérvalo [f1, f2].

- **betweenDays($f1, $days)**: comprueba si la fecha del objeto se encuentra entre el intérvalo [f1, (f1 + days)]. Si el número de dias es negativo se usa el intervalo [(f1 + days), f1]
- **getTimestamp()**: obtiene la fecha en timestamp. Este tipo tiene restricciones (en PHP < 5.3 el rango va de 1970 al 2036 aprox.), por lo que se recomienda no usar (casi todas las operaciones con timestamp se pueden hacer con DateTime).

3.7.3.3. gvHidraDate, gvHidraDatetime y gvHidraTime

gvHidraTime es la clase utilizada para indicar un campo tipo hora. Con disponibilidad de los siguientes métodos propios:

- • **setShowTimer(bool)**: Indica si se quiere o no mostrar el botón tooltip asociado al campo que mostrará el timepicker.
- **setIUemergente(bool)**: Indica si al situarnos en el campo se mostrará o no el calendario.

gvHidraDate y gvHidraDatetime son las clases usadas para indicar que el tipo de un campo es una fecha, con la diferencia que la segunda admite también la hora. Al definir un campo de este tipo, el framework ya se encarga de hacer las transformaciones necesarias de formatos, asi como de aplicar máscaras y calendarios para la edición.

Estas clases también disponen de los siguientes métodos propios:

- • **setCalendar(bool)**: indica si se muestra o no el calendario.
- **setDayOfWeek({none|short|long})**: indica si se quiere mostrar el dia de la semana.
- **setDayOfYear(bool)**: indica si se quiere mostrar el día del año.
- **setIUemergente(bool)**: Indica si al situarnos en el campo se mostrará o no el calendario.

Si no se hace uso de los métodos los valores por defecto son *calendario=false, dayOfWeek=none, dayOfYear=false.*

Hay que tener precaución con no usar el tipo gvHidraDate si el campo asociado tiene información de hora en la base de datos, ya que el framework produciria una excepción. Este comportamiento se ha definido de esta forma para evitar pérdidas de datos por truncamiento.

Nota: Hay que tener en cuenta que el tipo gvHidraTimestamp (que hereda de DateTime de PHP) es una marca de tiempo, con lo que vamos a trabajar con instantes de tiempo. En el caso de operaciones con fechas, al crear instancias de dicha clase el valor que coge por defecto es "now" (el instante de tiempo de la invocación); por lo que si se está trabajando con fechas esto puede acarrear errores. Para obtener la referencia al día actual se debe invocar al constructor con el parámetro "today". A continuación mostramos unos ejemplos:

```
// Ejemplo, instante actual 24/07/2009 13:28:30

$f = new gvHidraTimestamp();
print($f->format("d/m/Y H:i:s")); // equivalente a $f->formatUser()
//Resultado = 24/07/2009 13:28:30

$f = new gvHidraTimestamp("today");
print($f->format("d/m/Y H:i:s"));
//Resultado = 24/07/2009 00:00:00
```

```
$f = new gvHidraTimestamp("yesterday");
print($f->format("d/m/Y H:i:s"));
//Resultado = 23/07/2009 00:00:00
```

3.7.3.4. Ejemplos

Obtener información de una fecha:

```
$fpeticionIni = $objDatos->getValue('fpeticion');
if (is_null($fpeticionIni))
   return;
$day   = $fpeticionIni->format('d');
$month = $fpeticionIni->format('m');
$year  = $fpeticionIni->format('Y');
```

Comparar fecha:

```
$fpeticionIni = $objDatos->getValue('fpeticion');
if (is_null($fpeticionIni))
   return;
// es fecha futura?
if (new gvHidraTimestamp("today") < $fpeticionIni)
```

Asignar una fecha cualquiera:

```
$f = new gvHidraTimestamp();
$f->setDate(1950,12,24);
$f->setTime(15,0,0);
$objDatos->setValue("fsolucion", $f);
```

Modificar una fecha recibida:

```
$fpeticionIni = $objDatos->getValue('fpeticion');
if (is_null($fpeticionIni))
   return;
$fpeticionIni->subMonths(10);
$objDatos->setValue("fpeticion", $fpeticionIni);
```

Obtener la diferencia en dias entre dos fechas:

```
$diffDias = round(($fini->getTimestamp() - $ffin->getTimestamp()) /
 (24*60*60)) ;
// a partir de PHP 5.3
$intervalo = $fini->diff($ffin);
$diffDias = $intervalo->format('%a');
```

Usar la fecha en una sentencia sql de la conexión correspondiente:

```
$fechabd = $this->getConnection()->prepararFecha($fpeticionIni);
$sentencia = "update facturas set fecha_entrada = '$fechabd' where usuario =
 'xxx'";
```

Obtener un objeto fecha de la base de datos de una consulta que no hace el framework:

```
$this->consultar("SELECT fecha,fechahora from tabla" ,

 array( 'DATATYPES'=>array('fecha'=>TIPO_FECHA,'fechahora'=>TIPO_FECHAHORA)));
```

3.7.4. Números

3.7.4.1. Entendiendo los números en gvHidra

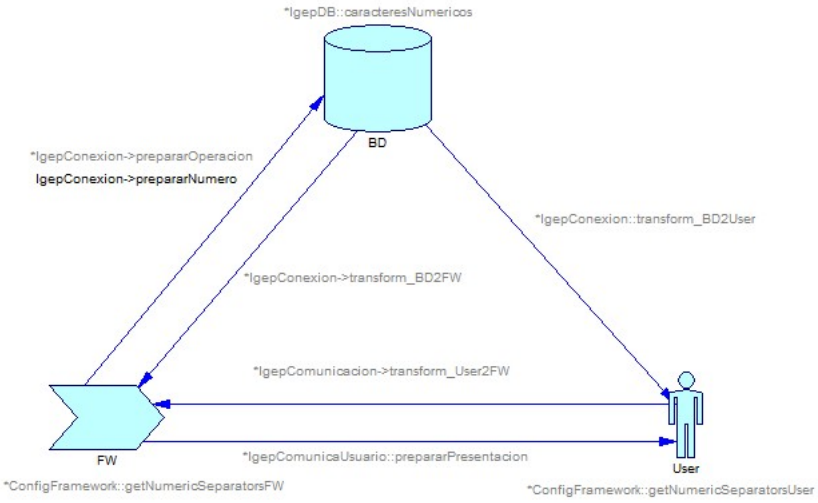
Cuando trabajamos con números podemos encontramos con los mismos problemas que hemos visto para las fechas, y que son los que vienen derivados de los distintos formatos que pueden usarse para representarlos.

En gvHIDRA se definen los formatos numéricos indicando el caracter usado como separador decimal, y el separador de miles (puede ser cadena vacia). Al igual que en las fechas, tenemos:

- formato de interfaz, el usado para interactuar con el usuario, y que viene definido en método **ConfigFramework::getNumericSeparatorsUser**.
- formato de datos, definido para cada tipo de gestor de base de datos, en métodos **caracteresNumericos** de **IgepDBMS_***. Es el formato usado para las operaciones con las bases de datos de ese tipo.
- formato del framework (o negocio), es el que el programador usa internamente para las operaciones, y se define en **ConfigFramework::getNumericSeparatorsFW**. Este formato coincide con la representación de números interna en PHP, es decir, con punto decimal y sin separador de miles. Por tanto el programador puede manejar los números directamente con los operadores y funciones propios de PHP.

El framework se encarga de hacer las conversiones necesarias en cada operación. De los tres formatos definidos, solo el de la interfaz es configurable por el programador.

En la siguiente figura podemos ver un esquema de la situación, siguiendo las mismas reglas explicadas anteriormente para las fechas:



3.7.4.2. gvHidraInteger y gvHidraFloat

- **gvHidraInteger**.

  Usaremos este tipo para datos de tipo entero. No dispone de métodos propios.

- **gvHidraFloat**.

  Para datos de tipo numérico con coma flotante. Métodos propios:

- **setFloatLength(int)**. Permite fijar la longitud de la parte decimal del número. De este modo el número se define a partir de su longitud total más la de la parte decimal (siguiendo el patrón de SQL). Si el número de decimales es 0, se recomienda usar gvHidraInteger.

  Nota 1: si no se hace uso del método setFloatLength el valor por defecto es 2.

  Nota 2: Si no coincide el número de decimales en la BD con los que le indicamos al tipo se produce una excepción, por lo que en caso necesario el programador tiene que truncar/redondear los datos según sus necesidades.

3.7.4.3. Uso en el framework

En el constructor de la clase manejadora del panel hay que crear el tipo para poder asignárselo al campo que corresponda, así como marcar las propiedades/características.

```
// numero con 8 digitos en la parte entera y 2 decimales
$tipoNumeroDec =  new gvHidraFloat(false, 10);
// numero con 7 digitos en la parte entera y 3 decimales
$tipoNumeroDec->setFloatLength(3);
$this->addFieldType('ediCoste', $tipoNumeroDec);
```

Primero se crea un objeto de tipo float llamando a la clase gvHidraFloat, con el primer parámetro indicamos la obligatoriedad del campo, y con el segundo el tamaño máximo del campo, incluidos los decimales. Ahora ya podemos hacer uso del método **setFloatLength()**. Y por último se asigna el tipo al campo en concreto, en nuestro caso es "ediCoste" (alias del campo en la select de búsqueda o edición).

De esta forma, gvHidra se encarga de realizar las conversiones necesarias para comunicarse con la BD y el usuario no necesita preocuparse por si debe introducir como separador de decimales la coma o el punto. Los separadores de miles aparecen "solos" y el separador decimal aparece con la coma o el punto del teclado numérico. Cualquier otro caracter (letras, paréntesis, etc...) no pueden introducirse, se ignoran.

Si trabajando en la capa de negocio queremos asignar explícitamente un número para mostrarlo en pantalla, lo podemos hacer directamente con el metodo **setValue()** del objeto datos usando un numero en PHP. Si lo que queremos es coger un número (getValue del objeto datos), operar con él y asignarlo, también lo podemos hacer directamente. Si tras operar con él queremos hacer algo con la base de datos, usaremos el método prepararNumero de la conexión correspondiente:

```
$costebd = $this->getConnection()->prepararNumero($coste);
$sentencia = "update facturas set importe = $costebd where factura = 10";
```

3.7.5. Creación de nuevos tipos de datos

Puede ser interesante que, para ciertos casos puntuales, se requiera crear un tipo de datos concreto que nos garantice que los datos son correctos. El framework realizará las validaciones por nosotros y nos garantizará que el dato es válido. Para ello debemos crear una clase dentro de nuestra aplicación o custom que sea accesible. Esta clase debe heredar de uno de los tipos básicos de gvHidra o del tipo básico (gvHidraTypeBase) si sólo se quieren heredar las características básicas. Esta clase debe implementar la interfaz gvHidraType. Esto supone que el programador de la clase deberá implementar el método validate(). A continuación tenemos un ejemplo:

```
//Type que controla que el campo introducido corresponde con el año actual
class anyoActual extends gvHidraTypeBase implements gvHidraType
{
    public function __construct($required=false)
```

```
    {
        $maxLength = 4;
        parent::__construct($required,$maxLength);
    }//Fin de constructor

    public function validate($value)
    {
        if($value!=date('Y'))
           throw new Exception('No ha introducido el año actual.');
    }

}
```

3.8. Listas de datos sencillas

3.8.1. Listas

Las listas de opciones son de gran ayuda para los formularios de las aplicaciones. Nos podemos encontrar listas desplegables, listas de tipo radiobutton o listas con checkbox. Vamos a ver como es el uso de estos elementos en gvHidra.

En primer lugar, tenemos que distinguir entre listas estáticas o listas dinámicas. Básicamente la diferencia se encuentra en el origen de datos que las crea/rellena. Su implementación es bastante similar aunque con algunas diferencias. Vamos a describir lo que tienen en común los dos tipos de listas.

Empezaremos viendo lo más sencillo, que es incluir una lista en la plantilla (tpl). El plugin a utilizar es el **CWLista**, vamos a hacer hincapié en el parámetro es "*value*", el resto se puede ver su definición y uso en el Apéndice Documentación de Plugins. En el parámetro nos vendrán los datos que compondrán la lista en cuestión, la variable asignada siempre deberá tener la misma estructura, la que vemos en el ejemplo, siendo la palabra "defaultData_" reservada.

```
{CWLista nombre="codigoProvincia" textoAsociado="Provincia" editable="true"
 dataType=$dataType_claseManejadora.codigoProvincia
 value=$defaultData_claseManejadora.codigoProvincia}
```

Pasamos a la parte que le corresponde a la clase manejadora del panel. En ella debemos definir la lista mediante la clase **gvHidraList**, este método define el tipo y contenido de la lista, para ello se le pasan unos parámetros:

1. *Nombre del campo destino de la lista* (campo *obligatorio*). Será el mismo nombre que le hayamos puesto en la tpl al plugin CWLista (p.ej. "codigoProvincia")

2. *Cadena que identifica la fuente de datos* (campo *opcional*). Este campo pasa a ser obligatorio en el caso de las listas dinámicas.

3. *DSN de conexión* (campo *opcional*). Permite incluir un DSN de conexión para indicar una conexión alternativa a la propia del panel. Si no se indica ninguna se cogerá por defecto el DSN del panel. Por ejemplo, podemos tener un panel trabajando en PostgreSQKL y que un campo tenga una lista desplegable que lea los datos en MySQL.

A partir de la clase **gvHidraList()** podemos utilizar los siguientes métodos para acabar de perfilar la lista:

- **addOption($valor, $descripcion)**: Nos permite añadir opciones a la lista.

- **setSelected($valor)**: Con este método le indicaremos qué opción aparecerá seleccionada por defecto.

- **setMultiple($multiple)**: Pasándole un parámetro booleano (true/false) indicaremos si es o no una lista múltiple.
- **setSize($size)**: Indicaremos el número de elementos visibles cuando estamos en una lista que es múltiple. Por defecto tendrá un size=5.
- **setDependence($listasCamposTpl, $listasCamposBD,$tipoDependencia=0)**: Método que permite asigar dependencia en una lista, es decir, si tenemos una lista cuyos valores dependen del valor de otros campos, necesitamos indicarlo con este método.
  - *$listasCamposTpl*: Será un array que contiene la lista de campos de la tpl de los cuales depende la lista.Array que, indexado en el mismo orden que el anterior, realiza la correspondencia de los campos del array anterior con los de la Base de Datos.
  - *$listaCamposBD*: Será un array que, indexado en el mismo orden que el anterior, realiza la correspondencia de los campos de la tpl con los de la base de datos.
  - *$tipoDependencia*: Un entero con el que le indicaremos si es una dependencia fuerte->0 o débil->1 (si no tiene valor el campo dependiente lo ignora).

  Nota: No se pueden crear listas dependientes con clausulas group by, ya que gvHidra internamente modifica en cada caso el valor de la where y en estos casos produce un error. De momento se pueden resolver usando una subconsulta con el group by en el from.

  ```
  $listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
  $this->addList($listaProvincias);
  $listaMunicipios = new gvHidraList('codigoMunicipio','MUNICIPIOS');
  $listaMunicipios-
  >setDependence(array('codigoProvincia'),array('tcom_municipios.cpro'));
  $this->addList($listaMunicipios);
  ```

- **setRadio($radio)**: Pasándole un parámetro booleano a true tendremos una lista de radiobuttons.

Una vez hemos definido los datos que compondrán la lista y cómo la queremos, tenemos que añadir la lista al panel, para ello utilizamos el método **addList()** del panel.

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$this->addList($listaProvincias);
```

3.8.1.1. Listas estáticas

Como su propio nombre indica, el origen de datos de estas listas vendrá dado por un conjunto de valores indicados de forma estática. Por lo tanto tendremos que definir la lista únicamente en el constructor de la clase manejadora.

Primero crearemos la lista haciendo uso de la clase **gvHidraList()**, a la que se le pasa por parámetro el nombre del campo que le hemos dado en la tpl. Ahora ya podemos ir añadiendo las opciones que necesitamos, lo haremos con el método **addOption()**, nos permitirá añadir tantas opciones como se necesiten.

A continuación vemos un ejemplo para que sólo aparezcan las provincias pertenecientes a la Comunidad Valenciana:

```
$listaProvincias = new gvHidraList('codigoProvincia');
$listaProvincias->addOption('03','ALICANTE');
$listaProvincias->addOption('12','CASTELLON');
$listaProvincias->addOption('46','VALENCIA');
```

```
$listaProvincias->setSelected('12');
$this->addList($listaProvincias);
```

3.8.1.2. Listas dinámicas

En este caso, el conjunto de valores posibles se obtienen a partir de una fuente de datos, ya sea de una consulta a base de datos, una clase...

Tal y como se ha indicado al principio, el segundo parámetro indica la fuente que se va a utilizar para obtener el contenido. Estas fuentes se pueden definir a nivel general de gvHIDRA (consultas o tablas comunes) o a nivel particular de cada aplicación, teniendo en cuenta que prevalecen las definidas en la aplicación sobre las generales del framework, si se han definido con el mismo nombre.

Las definiciones de las listas dinámicas deben ser cargadas al arrancar la aplicación, por ello se deben cargar en el constructor del objeto que maneja el panel principal de la aplicación, es decir, en la clase **AppMainWindow.php**.

Vamos a ver como incluir la fuente de datos, debemos hacer uso del objeto de configuración de gvHidra y del método apropiado según la fuente que carguemos, que pueden ser de dos tipos: fuentes de datos SQL o clases.

- **List_DBSource**

  Esta ha sido la fuente de datos habitual para las listas en gvHIDRA. Básicamente consiste en cargar directamente una query que gvHidra se encargará de parametrizar (añadir filtros a la where) para obtener el resultado esperado.

  En este caso haremos uso del método **setList_DBSource()** donde definiremos la query correspondiente, teniendo en cuenta los siguientes puntos:

  - La consulta ha de seleccionar dos campos, uno se corresponderán con el valor que guardará el campo (le pondremos de alias 'valor' en la query), y el otro corresponderá con la información que visualice la opción de la lista en pantalla (este deberá tener el alias 'descripcion')

  - Por defecto la ordenación es por el campo 'descripcion asc', aunque podemos modificarla con la clausula ORDER BY.

```
class AppMainWindow extends CustomMainWindow
{
  public function AppMainWindow()
  {
    ...
    $conf = ConfigFramework::getConfig();

    //Tipos
    $conf->setList_DBSource('TIPOS',"select ctipo as \"valor\", dtipo as
 \"descripcion\" from tinv_tipos");

    //Subtipos
    $conf->setList_DBSource('SUBTIPOS',"select cstipo as \"valor\", dstipo as
 \"descripcion\" from tinv_subtipos");

    //Estados
    $conf->setList_DBSource('ESTADOS',"select cestado as \"valor\", destado as
 \"descripcion\" from tinv_estados");
```

```
    ...
  }
}
```

En el ejemplo anterior vemos como con el método **setList_DBSource()** añadimos la definición de las fuentes de datos que se gastarán en la aplicación para cargar las listas. Este método tiene dos parámetros, el primero es un identificador de la lista, este identificador debe coincidir con el identificador que se le da a la definición de la lista en la clase manejadora (gvHidraList('nombreCampo','*TIPOS*');); y el segundo es la definición de la consulta SQL tal y como se ha indicado antes, con los alias correspondientes.

- **List_ClassSource**

Estas nos permiten definir como fuente de datos el resultado de un método de una clase. Esta clase, debe implementar la interfaz **gvHidraList_Source** para que el framework pueda interactuar con ella. Consiste en implementar el método *build* que será llamado por el framework para obtener el resultado de la consulta. Aquí introducimos un ejemplo sencillo:

```
/**
 * Fuente de datos ejemplo
 *
 * $Revision: 1.3.2.37 $
 */

class ejemploSource implements gvHidraList_Source
{
    public function __construct()
    {

    }

    public function build($dependence,$dependenceType)
    {

       $resultado = array(
                    array('valor'=>'01','descripcion'=>'UNO'),
                    array('valor'=>'02','descripcion'=>'DOS'),
                    array('valor'=>'03','descripcion'=>'TRES'),
                );
       return $resultado;
    }
}
```

De este ejemplo cabe destacar:

- La clase implementa la interfaz **gvHidraList_Source**. Si no implementa esta interfaz, no puede ser admitida como fuente de datos de las listas.

- El método **build**, devuelve como resultado un *array*. Este array, puede ser array vacio o un array que contiene por cada una de sus posiciones un array con dos índices válidos (valor y descripción).

Al igual que en el caso de las fuentes de datos tipo List_DBSource(), para incluirlas debemos hacer uso del objeto de configuración de gvHidra llamando al método apropiado.En este caso, el método a utilizar es setList_ClassSource. A continuación mostramos algunos ejemplos de carga.

```
class AppMainWindow extends CustomMainWindow
```

```
{
  public function AppMainWindow()
  {
    ...
    $conf = ConfigFramework::getConfig();

    $conf->setList_ClassSource('EJEMPLO','ejemploSource');

    //Subtipos
    $conf->setList_ClassSource('SUBTIPOS',"SubTipos");

    //Estados
    $conf->setList_DBSource('ESTADOS',"EstadosSource");
    ...
  }
}
```

En el ejemplo anterior vemos como con el método **setList_ClassSource()** añadimos la definición de las fuentes de datos que se gastarán en la aplicación para cargar las listas. Este método tiene dos parámetros, el primero es un identificador de la lista, este identificador debe coincidir con el identificador que se le da a la definición de la lista en la clase manejadora (gvHidraList('nombreCampo','*TIPOS*');); y el segundo es el nombre de la clase que actuará como fuente.

Una vez conociendo como funcionan tanto las listas estáticas como las dinámicas, es muy común que se utilice una mezcla de ambas. Este uso nos será útil, por ejemplo, en el siguiente caso:

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$listaProvincias->addOption('00','Todas');
$listaProvincias->setSelected('00');
$this->addList($listaProvincias);
```

En el ejemplo añadimos una opción que implica la selección de todas las opciones, hay que tener en cuenta estos valores que se añaden de forma estática, porque al interactuar con la base de datos ese valor no existirá, por lo tanto habrá que añadir un control particular en la clase manejadora.

Otro ejemplo puede ser que el campo de la base de datos admita también nulos, así que tendremos que añadir una opción de valor nulo y descripción en blanco, para darnos la posibilidad de no añadir ningún valor:

```
$listaProvincias->addOption('',' ');
```

3.8.1.3. Listas Dependientes

Puede darse el caso de que el contenido de la lista sea dependiente de otros campos o listas. Para ello tendremos que cargar los valores dependiendo de otro valor.

Nota

El uso de listas dependientes aumenta el proceso de cálculo de nuestras ventanas ya que lanzará una consulta por cada registro que obtengamos. Para optimizar consultar método setLazyList

Para conseguir listas dependientes, en la definición de las listas indicamos cual es la dependencia, mediante el método setDependencia de la clase gvHidraList. Siguiendo con nuestro ejemplo de provincias, es

habitual encontrarnos con el par de listas provincias/municipios de forma que al seleccionar una provincia recargue la lista con los municipios de dicha provincia seleccionada.

```
$listaProvincias = new gvHidraList('codigoProvincia','PROVINCIAS');
$this->addList($listaProvincias);

$listaMunicipios = new gvHidraList('codigoMunicipio','MUNICIPIOS');
$listaMunicipios->setDependence(array('codigoProvincia'),
                  array('tcom_municipios.cpro'));
$this->addList($listaMunicipios);
```

Como podemos ver, se hace una llamada al método setDependence en la lista de municipios donde se le pasan dos valores. El primero corresponde con el nombre del campo de la Tpl de la lista "padre"; es decir de la lista de la que dependemos; el segundo es el nombre de dicho campo en la consulta que define la lista. El resto del código corresponde al de una definición normal de una lista. Tiene un tercer parámetro para indicar el tipo de dependencia (fuerte->0, débil->1). Por defecto es fuerte (el valor de la lista dependiente dependerá siempre del valor de la otra lista).

Nota

No se pueden crear listas dependientes con clausulas group by, ya que gvHidra internamente modifica en cada caso el valor de la where y en estos casos produce un error. De momento se pueden resolver usando una subconsulta con el group by en el from.

El siguiente paso es la definición en la Tpl. Para ello haremos uso del componente CWLista pero con una serie de parámetros especiales. Concretamente debemos indicar que el componente CWLista de provincias actualiza al de municipios.

```
{CWLista nombre="codigoProvincia" size="3"
 actualizaA="codigoMunicipio" editable="true"
 textoAsociado="Provincia"
 dataType=$dataType_claseManejadora.codigoProvincia datos=
$defaultData_claseManejadora.codigoProvincia}

{CWLista nombre="codigoMunicipio" size="3" editable="true"
 textoAsociado="Municipio"
      dataType=$dataType_claseManejadora.codigoMunicipio}
```

3.8.1.4. Uso en acciones

Las listas tienen una serie de métodos especiales que permiten su manipulación en las acciones. Para acceder a una lista desde una acción de este tipo se debe hacer uso de los siguientes métodos:

- **getLista**:obtenemos la lista con la que queremos trabajar.
- **clean**: limpiamos los valores dejándola vacia.
- **setLista**: fijamos una lista.
- **setSelected**: marcamos un valor como seleccionado.
- **getDescription**: devuelve la descripción del valor seleccionado.
- **getSelected**: devuelve el valor seleccionado.

- **toArray**: obtenemos una lista en formato array.
- **arrayToObject**: cargamos un objeto lista con un array.

A continuación mostramos un ejemplo de uso:

```
$lcuentaFe = $objDatos->getList('cuenta_fe');
$lcuentaFe->clean();

$lcuentaFe->addOption('','');

if (count($datos)>0) {

 foreach($datos as $datos_lista) {
     //valor, descripcion
     $lcuentaFe->addOption($datos_lista['valor'],$datos_lista['descripcion']);
    }
}
$lcuentaFe->setSelected('');

$objDatos->setList('cuenta_fe',$lcuentaFe);
...
```

3.8.2. Checkbox

El elemento checkbox podemos emplearlo también para una lista de valores, nos permitirá elegir más de una opción ya que no serán excluyentes. Para definir el checkbox hay que hacer uso de la clase **gvHidra-CheckBox** y de sus metodos, que son:

- **setValueChecked($value)**: Fija el valor que se quiere tener cuando el check está marcado.
- **getValueChecked()**: Devuelve el valor que tiene el check cuando está marcado.
- **setValueUnchecked($value)**: Fija el valor que se quiere tener cuando el check está desmarcado.
- **getValueUnchecked()**: Devuelve el valor que tiene el check cuando está desmarcado.
- **setChecked($boolean)**: Indicar si el checkbox aparece marcado o no por defecto.

De este modo en la clase manejadora tendremos que crear el checkbox y definir las propiedades:

```
//Creamos checkbox y lo asociamos a la clase manejadora
$filCoche = new gvHidraCheckBox('filCoche');
$filCoche->setValueChecked('t');
$filCoche->setValueUnchecked('f');
$this->addCheckBox($filCoche);
```

En la tpl definiremos el checkbox de la siguiente forma:

```
{CWCheckBox nombre="filCoche" editable="true" textoAsociado="Coche" dataType=
$dataType_claseManejadora.filCoche
valor=$defaultData_claseManejadora.filCoche}
```

Es importante añadir el parámetro dataType que es el que asociará la definición dada en la clase manejadora con el plugin.

3.9. Mensajes y Errores

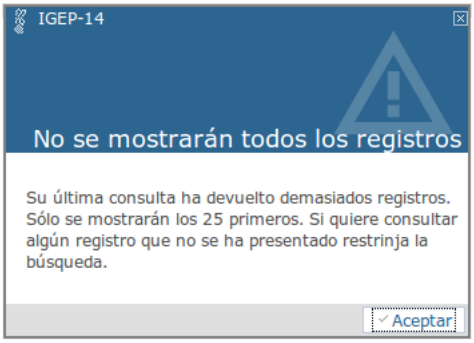
En cualquier aplicación es necesario mostrar avisos, errores... En gvHidra todo este tipo de mensajes se ha clasificado en cuatro grupos, más que nada por su aspecto visual, ya que todas serán invocadas de la misma forma, asociando un color a un tipo de mensaje:

1. **Alertas**.

   Este tipo de mensajes será útil para alertar al usuario de un estado, aunque pueda continuar trabajando pero con conocimiento de un estado. En el ejemplo se alerta de que la aplicación está en desarrollo por lo tanto puede ser inestable.

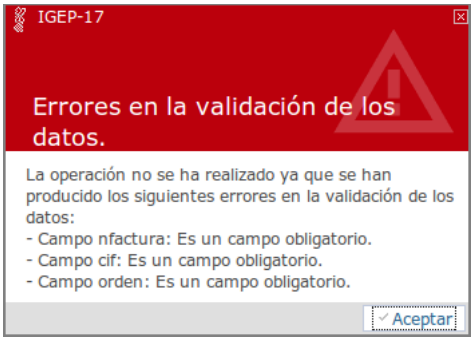
2. **Avisos**.

   Este tipo de mensaje nos avisa de cierta situación, nada problemática, pero que tengamos en cuenta.



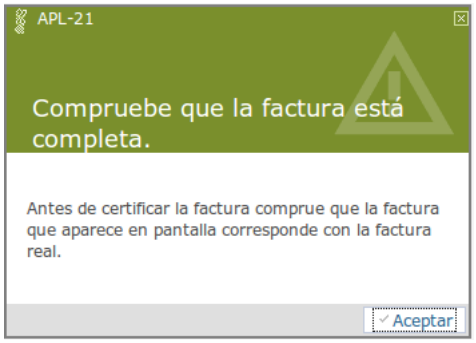
3. **Errores**.

   Claramente los errores muestran problemas ocurridos al efectuar una acción. En el ejemplo se avisa de que hay que rellenar ciertos campos de forma obligatoria y por eso falla la búsqueda.



4. **Sugerencias**.

Son mensajes de tipo consejo al usuario que no le impiden continuar con el trabajo. En el ejemplo se aconseja al usuario cerciorarse de que todo esté rellenado antes de efectuar la acción.



Hay *mensajes que son propios del framework*, son mensajes generales a cualquier aplicación, se encuentran ubicados en la clase **IgepMensaje.php**. Por otro lado estarán los *mensajes particulares de cada aplicación*, estos se definirán en el fichero **mensajes.php**, este fichero se encuentra en el directorio raíz de la aplicación.

Vamos a explicar como añadir mensajes en el fichero *mensajes.php*. En este fichero existe una variable global que es el array donde se irán almacenando los mensajes (**$g_mensajesParticulares**).

```
<?php

global $g_mensajesParticulares;
$g_mensajesParticulares = array(
     'APL-1'=>array('descCorta'=>'No se puede realizar el
 borrado','descLarga'=>'No se puede borrar un tipo que tiene subtipos
 asociados. Si quiere eliminar este tipo deberá borrar todos sus
 subtipos.','tipo'=>'ERROR'),
     'APL-2'=>array('descCorta'=>'Se ha listado la factura.','descLarga'=>'Se
 ha listado la factura %0%-%1%.','tipo'=>'AVISO'),
...
);

?>
```

Un mensaje se crea añadiendo un elemento al array asociativo. El elemento tiene una clave única (ej. 'APL-1'), ya que esta clave es la que se utilizará como identificador para invocarlo, y como valor es otro array que contiene tres elementos:

1. **descCorta**: Descripción corta del mensaje, esta descripción aparecerá en la parte superior del mensaje, en la zona coloreada.

2. **descLarga**: Descripción completa del mensaje, esta descripción aparecerá en la parte inferior del mensaje, zona blanca. Aquí podemos jugar con el texto del mensaje y pasarle parámetros desde su invocación, nos dará un mensaje más personalizado. Los valores vendrán en un array, y aquí se hará referencia a ellos de la siguiente forma %0% para el primer valor del array, %1% para el segundo, y así sucesivamente.

3. **tipo**: palabra clave que definirá el tipo del mensaje. Estas palabras pueden ser: AVISO, ERROR, SUGERENCIA y ALERTA, que se corresponden con los cuatro grupos vistos anteriormente.

La invocación de los mensajes se puede efectuar desde dos puntos distintos, desde código o mediante parámetro del plugin.

3.9.1. Invocación desde código

Es el uso más habitual. Para invocar un mensaje hay que hacer uso del método **showMessage()** y pasándole como parámetro el identificador del mensaje que queremos mostrar, que debe coincidir con el definido en *mensajes.php*:

```
if ( <condicion> ) {
   $this->showMessage('APL-3');
   return 0;
}
```

También podemos pasar argumentos a los mensajes, atributos para personalizar mejor el mensaje que queremos dar. En este caso, en el texto del mensaje deberemos colocar las variables a sustituir, de la forma '%0%', '%1%', ... tal y como hemos explicado anteriormente. Estas variables se pasan en el método **showMessage()** mediante un array con tantos valores como variables hayamos definido. A continuación se muestra un ejemplo:

```
// APL-4: El valor ha de ser mayor de %0% y menor de %1%
if ( <condicion> ) {
    $this->showMessage('APL-4', array('5','10'));
    return 0;
}
```

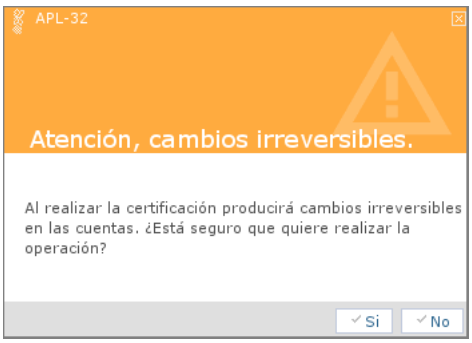
3.9.2. Invocación como confirmación

En este caso nos referimos a ventanas que solicitan del usuario una confirmación para continuar con la acción o cancelarla.

Para crear este tipo de mensajes de confirmación hay que añadir el parámetro "*confirm*" al plugin **CW-Boton** (ver Apéndice X Documentación plugins) indicando el identificador del mensaje que queremos mostrar (ej. "APL-2").

De este modo, nos aparecerá un mensaje con dos alternativas Si/No. Al pulsar Si se ejecutará la acción, al pulsar No, la acción quedará cancelada.

```
{CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"
 confirm="APL-32"}
```



3.10. Uso de datos por defecto

En este apartado hablaremos del uso de los datos por defecto en gvHIDRA. El framework tiene un conjunto de datos reservados para ser mostrados en los casos en los que la visualización de pantalla no tenga el respaldo de datos. Es decir, el framework muestra datos por defecto sólo en los casos que no tengamos datos de una fuente de datos. Estos casos son:

1. *Inserción:* es el caso más habitual. Cuando pasamos al modo de trabajo inserción y se visualizan los campos para que el usuario inserte la nueva información, el framework hace uso de la información de datos por defecto.

2. *Filtro de búsqueda:* Cuando estamos en el modo filtro, el framework hace uso de los datos por defecto para poder fijar unos parámetros iniciales. Esto puede ser util para fijar, por ejemplo, el año en curso como valor básico para el campo año.

Pero, ¿Cómo podemos hacerlo? Bien, el framework proporciona un método para poder fijar estos valores. Se trata del método addDefaultData que admite dos parámetros: el nombre del campo y el valor.

Por otro lado, en la tpl debemos indicar al componente que haga caso al valor por defecto. Para ello debemos indicarle en la propiedad valor que lea el valor de la variable del framework defaultData indicando el nombre de la clase Manejadora. En el siguiente ejemplo fijaremos el valor 2012 al campo fil_anyo para la claseManejadora Presupuesto.

```
//en la clase manejadora

$this->addDefaultData('fil_anyo','2012');

...

{*en la tpl*}
{... value=$defaultData_Presupuesto.fil_anyo...}
```

Capítulo 4. Elementos de pantalla avanzados

4.1. Patrones complejos

4.1.1. Maestro/Detalle

4.1.1.1. Maestro/Detalle

Un maestro-detalle es el caso de una ventana con paneles dependientes, este caso no se diferencia mucho de una ventana normal de gvHidra aunque tiene sus peculiaridades.

Al igual que en las ventanas "normales", en primer lugar se debe crear una clase por cada uno de los paneles que aparezca en la pantalla, es decir, una clase para el maestro y una para el detalle. Como siempre estas clases se depositarán en el directorio actions de la estructura de la aplicación.

Las clases, tanto la del maestro como la del detalle, son como cualquier clase de un panel, tal y como se ha explicado en el punto 3.2.2 Paso a paso, salvo unas anotaciones:

- *Clase manejadora del maestro*.

  Un panel que tenga otro/s panel/es dependiente/s tendrán que utilizar el método **addSlave()** en el constructor de la clase. A este método se le pasan tres parámetros, el primero es el nombre de la clase hija, el segundo y tercer parámetro son dos arrays que definirán por qué campos se establece la relación entre maestro y detalle. El segundo es un array con los campos del padre que van a ser utilizados para recargar el hijo, y el tercer parámetro, un array con el nombre de los campos del detalle que se corresponden con los del maestro.

```
class TinvTipos extends gvHidraForm_DB
{
   function TinvTipos()
   {
      ...
      $this-
>addSlave("TinvSubtipos",array("lisCodigoTipo"),array("ediCodigoTipo"));
      ...
   }
}
```

  Por ejemplo, tenemos que crear un mantenimiento para dos tablas: tipos y subtipos. Estas tablas tienen una relación 1:N (por cada tipo pueden haber N subtipos). Implementado en gvHidra, tendremos dos clases en el directorio actions (p.e. TinvTipos y TinvSubtipos). La clase *TinvTipos* tiene un hijo que se llama *TinvSubtipos*. El campo de la tpl *ediCodigoTipo* (en el panel maestro), se utiliza para identificar a los subtipos (los detalles) y en el panel detalle existe un campo que hace referencia al campo del maestro y cuyo nombre es lisCodigoTipo.

- *Clase manejadora del detalle*.

  Igual que en el maestro, el detalle también tiene que tener una referencia a su padre. Concretamente se debe utilizar el método **addMaster()** en el constructor de la clase, donde se le pasará como parámetro

el nombre de la clase que maneja el panel maestro. Por tanto, siguiendo con nuestro ejemplo, la clase TinvSubtipos debe incorporar la siguiente línea:

```
class TinvSubtipos extends gvHidraForm_DB
{

     public function __construct()
     {
        ...
        $this->addMaster("TinvTipos");
        ...
     }
}
```

El siguiente cambio respecto a las ventanas "normales" viene en la creación de los paneles en los ficheros ubicados en el directorio *views*. Deberemos definir los dos paneles, maestro y detalle, a partir de la clase IgepPanel. Una vez definidos debemos utilizar el método de la clase **IgepPantalla, agregarPanelDependiente()**, para agregar el panel detalle al maestro. A este método se le han de pasar dos parámetros, el primero debe ser el panel a agregar y, el segundo, el nombre de la clase que maneja el panel padre. El fichero de views podría quedar así:

```
<?php
$comportamientoVentana = new IgepPantalla();

// Definición del MAESTRO
$panelMaestro = new IgepPanel('TinvTipos',"smty_datosTablaM");
$panelMaestro->activarModo("fil","estado_fil");
$panelMaestro->activarModo("lis","estado_lis");
$comportamientoVentana->agregarPanel($panelMaestro);

// Definición del DETALLE
$panelDetalle = new IgepPanel('TinvSubtipos',"smty_datosFichaD");
$panelDetalle->activarModo("edi","estado_edi");
// Agregamos el panel detalle al maestro
$comportamientoVentana->agregarPanelDependiente($panelDetalle,"TinvTipos");

$s->display('tablasMaestras/p_tiposubtipo.tpl');
?>
```

Finalmente, en la plantilla (tpl) para un maestro detalle también existen características a destacar de una plantilla para una ventana sencilla. Vamos a ver los puntos:

- Hay que indicar que es un panel maestro con el atributo **esMaestro** del plugin **CWPanel** (esMaestro="true")
- El parámetro **itemSeleccionado** del plugin **CWPanel** es un parámetro interno para el correcto funcionamiento de gvHidra, por eso se asigna a una variable smarty, **itemSeleccionado=$smty_filaSeleccionada**, para mantener el maestro seleccionado.
- El botón tooltip (**CWBotonTooltip**) para insertar registros en el maestro tiene un parámetro **ocultarMD** con el que le indicamos qué panel se debe ocultar cuando pasamos a modo inserción, ya que va a ser un registro nuevo del maestro por lo tanto aún no tiene detalle.
- En el plugin **CWContenedorPestanyas** se tiene que indicar con el parámetro **id** si las pestañas son del maestro o del detalle.

- Las pestañas del panel maestro (**CWPestanya**) tienen unos parámetros, **mostrar** y **ocultar**, con los que se indicará si el panel detalle lo queremos visible o no. Además está el parámetro **panelAsociado** donde se indica a qué panel se refiere esa pestaña, ya que vamos a tener pestañas tanto para el maestro como para el detalle.

- Llegamos a la parte de definir el detalle, establecemos una condición con sintaxis de smarty **{if count($smty_datosTablaM) gt 0}**, para que no nos muestre el panel detalle si no hay datos en el maestro.

- El identificador, **id**, que le demos al panel del detalle (**CWPanel**) deberá ser lisDetalle, en el caso de estar en un modo tabular, o ediDetalle, en modo registro. También hay que indicar el parámetro **detalleDe** pasándole el **id** del maestro.

- Tanto si el detalle es una ficha (**CWFichaEdicion**) como un tabular (**CWTabla**) hay que definir su **id**, *FichaDetalle* o *TablaDetalle*, respectivamente.

- En el panel del detalle es necesario definir el o los campos que contengan la clave del maestro. En el siguiente ejemplo tenemos un maestro-detalle, siendo la clave del maestro el campo "*lisCodigoTipo*", el detalle necesita del valor de este campo, para ello vemos que en el panel del detalle tenemos un campo "*ediCodigoTipo*", que será el que nos relacione el detalle con su maestro. Para que el campo del detalle tenga el valor de la clave del maestro hay que definir su propiedad value, en el ejemplo **value=$defaultData_TinvSubtipos.ediCodigoTipo**.

  Si nos encontramos en el caso de un detalle "Tabular-Registro" hay que definir estos campos clave tanto para el panel tabular como para el panel registro.

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=
$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
{CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=
$smty_customTitle}
  {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
{/CWBarra}

{CWMarcoPanel conPestanyas="true"}

<!-- ********************** PANEL MAESTRO **********************-->
<!--*********** PANEL fil *****************-->
  {CWPanel id="fil" action="buscar" method="post" estado=$estado_fil
 claseManejadora="TinvTipos"}
     {CWBarraSupPanel titulo="Tipos de Bienes"}
        {CWBotonTooltip imagen="04" titulo="Limpiar Campos" funcion="limpiar"
 actuaSobre="ficha"}
     {/CWBarraSupPanel}
     {CWContenedor}
        {CWFicha}
              <br />
              {CWCampoTexto nombre="filCodigoTipo"
 editable="true" size="2" textoAsociado="Código de Tipo" dataType=
$dataType_TinvTipos.filCodigoTipo}
            <br /><br />
              {CWCampoTexto nombre="filDescTipo"
 editable="true" size="30" textoAsociado="Descripción de Tipo" dataType=
$dataType_TinvTipos.filDescTipo}
            <br /><br />
        {/CWFicha}
```

```
     {/CWContenedor}
     {CWBarraInfPanel}
      {CWBoton imagen="50" texto="Buscar" class="boton" accion="buscar"}
     {/CWBarraInfPanel}
  {/CWPanel}

<!--********** PANEL lis *****************-->
  {CWPanel id="lis"
 tipoComprobacion="envio" esMaestro="true" itemSeleccionado=
$smty_filaSeleccionada action="operarBD" method="post" estado=$estado_lis
 claseManejadora="TinvTipos"}
     {CWBarraSupPanel titulo="Tipos de Bienes"}
        {CWBotonTooltip imagen="01" titulo="Insertar registros"
 funcion="insertar" actuaSobre="tabla" ocultarMD="Detalle"}
        {CWBotonTooltip imagen="02" titulo="Modificar registros"
 funcion="modificar" actuaSobre="tabla"}
        {CWBotonTooltip imagen="03" titulo="Eliminar registros"
 funcion="eliminar" actuaSobre="tabla"}
          {CWBotonTooltip imagen="04" titulo="Limpiar Campos"
 funcion="limpiar" actuaSobre="tabla"}
     {/CWBarraSupPanel}
     {CWContenedor}
      {CWTabla conCheck="true" conCheckTodos="false" id="Tabla1"
 numFilasPantalla="4" datos=$smty_datosTablaM}
         {CWFila tipoListado="false"}
            {CWCampoTexto nombre="lisCodigoTipo" editable="nuevo" size="2"
 textoAsociado="Cod.Tipo" dataType=$dataType_TinvTipos.lisCodigoTipo}
            {CWCampoTexto nombre="lisDescTipo" editable="true" size="25"
 textoAsociado="Tipo" dataType=$dataType_TinvTipos.lisDescTipo}
         {/CWFila}
         {CWPaginador enlacesVisibles="3"}
      {/CWTabla}
     {/CWContenedor}
     {CWBarraInfPanel}
      {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
      {CWBoton imagen="42" texto="Cancelar" class="boton" accion="cancelar"}
     {/CWBarraInfPanel}
  {/CWPanel}

<!-- ****************** PESTAÑAS ************************-->
  {CWContenedorPestanyas id="Maestro"}
     {CWPestanya tipo="fil" panelAsociado="fil" estado=
$estado_fil ocultar="Detalle"}
     {CWPestanya tipo="lis" panelAsociado="lis" estado=
$estado_lis mostrar="Detalle"}
  {/CWContenedorPestanyas}
</td></tr>
<tr><td>
<!-- ****************** PANEL DETALLE ************************-->
  {if count($smty_datosTablaM) gt 0}
   {CWPanel id="ediDetalle" tipoComprobacion="envio"
 action="operarBD" detalleDe="lis" method="post" estado="on"
 claseManejadora="TinvSubtipos"}
```

```
   {CWBarraSupPanel titulo="Subtipos de Bienes"}
      {CWBotonTooltip imagen="01" titulo="Insertar registros"
 funcion="insertar" actuaSobre="ficha"}
      {CWBotonTooltip imagen="02" titulo="Modificar registros"
 funcion="modificar" actuaSobre="ficha"}
      {CWBotonTooltip imagen="03" titulo="Eliminar registros"
 funcion="eliminar" actuaSobre="ficha"}
   {/CWBarraSupPanel}
   {CWContenedor}
      {CWFichaEdicion id="FichaDetalle" datos=$smty_datosFichaD}
         {CWFicha}
            <br /><br />
            {CWCampoTexto nombre="ediCodigoSubtipo" editable="nuevo" size="3"
 textoAsociado="Código" dataType=$dataType_TinvSubtipos.ediCodigoSubtipo}
            <br /><br />
            {CWCampoTexto nombre="ediDescSubtipo" editable="true" size="35"
 textoAsociado="Descripción" dataType=$dataType_TinvSubtipos.ediDescSubtipo}
            <br /><br />
            {CWCampoTexto nombre="ediCodigoTipo" oculto="true" value=
$defaultData_TinvSubtipos.ediCodigoTipo}
         {/CWFicha}
         {CWPaginador enlacesVisibles="3"}
      {/CWTabla}
   {/CWContenedor}
   {CWBarraInfPanel}
      {CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
      {CWBoton imagen="42" texto="Cancelar" class="boton" accion="cancelar"}
   {/CWBarraInfPanel}
  {/CWPanel}
<!-- ****************** PESTAÑAS ************************-->
  {CWContenedorPestanyas id="Detalle"}
   {CWPestanya tipo="lis" panelAsociado="ediDetalle" estado=$estado_edi}
  {/CWContenedorPestanyas}
 {/if}
{/CWMarcoPanel}
{/CWVentana}
```

Una vez tenemos claro como definir los ficheros correspondientes a la pantalla, vamos a comentar un aspecto del **mappings.php** respecto a un maestro/detalle. Respecto al maestro, hay que añadir una entrada que permita recargar el detalle a partir del registro seleccionado en el maestro, para ello se deben agregar las siguientes líneas en el mappings.php para la clase del maestro:

```
$this->_AddMapping('TinvTipos__recargar', 'TinvTipos');
$this->_AddForward('TinvTipos__recargar', 'gvHidraSuccess', 'index.php?
view=views/patronesMD/M(FIL-LIS)-D(LIS)/p_tiposubtipo.php&panel=listar');
$this->_AddForward('TinvTipos__recargar', 'gvHidraError', 'index.php?
view=views/patronesMD/M(FIL-LIS)-D(LIS)/p_tiposubtipo.php&panel=listar');
```

4.1.1.1.1. Optimización

Existe una combinación que aumenta de forma exponencial el proceso de cálculo de nuestras aplicaciones: el uso de listas dependientes en maestros detalles. Para solventar este problema, gvHIDRA desde su versión 4.0.0 incorpora un mecanismo de carga Lazy de las listas.

El sistema lanza una consulta por cada uno de los maestros recuperados, aunque, si estamos utilizando un maestro registro, sólo podemos visualizar uno cada vez. Para mejorar el rendimiento en estos casos, podemos utilizar el método setLazyList.

```
public function __construct() {
...
    $this->setLazyList(true);
}
```

Con la carga Lazy mejoramos el rendimiento pero sólo está disponible en los maestros detalle. Se recomienda sólo en el caso de los maestros en modo EDI (Registro)

4.1.1.2. Maestro/NDetalles

En este caso nos encontramos con un maestro con más de un detalle, visualmente tendremos un panel superior que corresponderá con el maestro, y en la parte inferior tendremos los detalles, estos serán accesibles mediante unas solapas superiores (ver imagen en el capítulo 2 Características técnicas).

En general, la forma de funcionar es muy parecida al maestro-detalle explicado en el punto anterior, salvo pequeñas diferencias. Igual que siempre, tenemos que definir una clase por cada panel, es decir, una por el maestro y otra por cada detalle que vayamos a tener. En este caso, en el maestro tenemos que añadir la referencia a todos los detalles que se vayan a tener con el método **addSlave()**.

```
class Expedientes extends gvHidraForm_DB
{
   function Expedientes()
   {
      ...
      $this-
>addSlave('informes',array('ediNexpediente'),array('lisNexpediente'));
      $this-
>addSlave('deficiencias',array('ediNexpediente'),array('lisNexpediente'));
      ...
   }
}
```

Los paneles detalle añadirán la referencia al padre igual que en el caso de un maestro-detalle simple, con el método **addMaster()**.

En el fichero de views tendremos que definir todos los paneles detalle que tengamos y agregarlos con el método **agregarPanelDependiente()**. Otra peculiaridad de este panel es que tenemos que definir las solapas que irán asociadas a los paneles detalle, para ello tenemos que definir un array asociativo con tantos arrays como detalles, estos deben tener dos claves, "*panelActivo*" y "*titDetalle*", que se corresponden, respectivamente, con el nombre de la clase manejadora del detalle y el texto que queramos que aparezca en la solapa. Otro punto a tener en cuenta es indicar qué detalle queremos que aparezca activo, para ello tenemos que asignar a la variable smarty *smty_panelActivo* el nombre de la clase manejadora que queramos.

```
<?php
//MAESTRO
$comportamientoVentana= new IgepPantalla();
$panelMaestro = new IgepPanel('expedientes',"smty_datosTablaM");
$panelMaestro->activarModo("fil","estado_fil");
$panelMaestro->activarModo("edi","estado_edi");
$datosPanel = $comportamientoVentana->agregarPanel($panelMaestro);

//DETALLE
```

```
$panelDetalle = new IgepPanel('informes',"smty_datosInformes");
$panelDetalle->activarModo("lis","estado_lis");
$datosPanelDetalle = $comportamientoVentana-
>agregarPanelDependiente($panelDetalle,"expedientes");

$panelDetalle = new IgepPanel('deficiencias',"smty_datosDeficiencias");
$panelDetalle->activarModo("edi","estado_edi");
$datosPanelDetalle = $comportamientoVentana-
>agregarPanelDependiente($panelDetalle,"expedientes");

// Botones detalles
// panelActivo: Nombre de la clase manejadora del detalle q vamos a activar
// titDetalle: Título q aparecerá en la pestaña del panel
$detalles = array (
      array (
            "panelActivo" =>"informes",
            "titDetalle" =>"Informes"
      ),
      array (
            "panelActivo" =>"deficiencias",
            "titDetalle" =>"Deficiencias"
      )
);
$s->assign('smty_detalles',$detalles);

$s->assign('smty_panelActivo','informes');

$s->display('MaestroNDetalles/p_expedientes.tpl');
?>
```

En la tpl también hay que señalar algunas diferencias respecto a un maestro-detalle simple. La parte que corresponde al maestro es exactamente igual que la explicada para el maestro-detalle, las diferencias las encontramos en la parte que corresponde a los detalles. Vamos a verlo por puntos:

- Vemos en la sección detalles se comprueba, mediante un if de smarty, si en el maestro hay datos. Si los hay pasamos a dibujar las solapas de los diferentes detalles con el plugin **CWDetalles**.
- A continuación englobamos cada panel detalle dentro de un if condicional que comprueba si el panel es el activo por defecto.

```
  1 {CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=
$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
    {CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=
$smty_customTitle}
    {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
    {/CWBarra}
  5
    {CWMarcoPanel conPestanyas="true"}

    <!-- ********************** PANEL MAESTRO **********************-->
    <!--*********** PANEL fil ******************-->
 10  {CWPanel id="fil" action="buscar" method="post" estado=$estado_fil
 claseManejadora="expedientes"}
     ...
```

```
     {/CWPanel}

    <!-- ****************** PANEL edi ***********************-->
 15  {CWPanel id="edi" tipoComprobacion="envio" esMaestro="true"
 itemSeleccionado=$smty_filaSeleccionada action="operarBD" method="post"
 estado="$estado_edi" claseManejadora="expedientes" accion=
$smty_operacionFichaexpedientes}
     ...
     {/CWPanel}

    <!-- ****************** PESTAÑAS ***********************-->
 20  {CWContenedorPestanyas id="Maestro"}
       {CWPestanya tipo="fil" panelAsociado="fil" estado=$estado_fil
 ocultar="Detalle"}
       {CWPestanya tipo="edi" panelAsociado="edi" estado=$estado_edi
 mostrar="Detalle"}
     {/CWContenedorPestanyas}

 25 <!-- ****************** PANELES DETALLES ***********************-->
    </td></tr>
    {if count($smty_datosTablaM ) gt 0 }
     {CWDetalles claseManejadoraPadre="expedientes" detalles=$smty_detalles
 panelActivo=$smty_panelActivo}
    <tr><td>
 30
    <!-- ****************** Detalle 1: INFORMES ***********************-->
     {if $smty_panelActivo eq "informes"}
       {CWPanel id="lisDetalle" tipoComprobacion="envio" action="operarBD"
 method="post" detalleDe="edi" estado="on" claseManejadora="informes"}
       ...
 35    {/CWPanel}
       {CWContenedorPestanyas  id="Detalle"}
          {CWPestanya tipo="lis" panelAsociado="lisDetalle" estado="on"}
       {/CWContenedorPestanyas}
     {/if}
 40
    <!-- ****************** Detalle 2: DEFICIENCIAS ***********************-->
     {if $smty_panelActivo eq "deficiencias" }
       {CWPanel id="ediDetalle" tipoComprobacion="envio" action="operarBD"
 method="post" detalleDe="edi" estado="on" claseManejadora="deficiencias"}
       ...
 45    {/CWPanel}
       {CWContenedorPestanyas  id="Detalle"}
          {CWPestanya tipo="edi" panelAsociado="ediDetalle" estado="on"}
       {/CWContenedorPestanyas}
     {/if}
 50
    {/if}
    {/CWMarcoPanel}
    {/CWVentana}
```

4.1.2. Árbol

Vamos a explicar ahora como desarrollar un mantenimiento tipo árbol, uno de los patrones más complejos. El aspecto visual del árbol es el siguiente:

La pantalla se divide claramente en dos zonas. La zona de la derecha se corresponde con lo que es el árbol en sí, y la de la izquierda corresponde con un panel común (tabular o registro). El usuario puede ir navegando por el árbol, expandiendo los nodos tal y como se haya especificado en el código, una vez llegado a la opción que nos interese en la parte derecha aparecerá un panel que visualizará los datos correspondientes al nodo seleccionado (p.e. en la imagen se puede ver que se muestra una factura del tercer trimestre para el año 1993, que es el nodo seleccionado).

4.1.2.1. ¿Cómo definir la estructura del árbol?

Primero hay que tener claro como va a ser la estructura del árbol, para poder definir los nodos y hojas del árbol. Estás definiciones, básicamente, se hacen declarando los "tipos" de nodos y como se obtienen sus hijos, indicando qué nodos aparecerán en la raiz, y los que vayan a ser hojas no tendrán especificación de sus hijos. De esta forma obtenemos un árbol muy flexible, en el que no hay que preocuparse de los niveles que va a tener.

Para definir un nodo (o nodo raiz) tenemos que especificar como se obtienen sus hijos, esto se puede hacer de dos modos:

- **SELECT**: Los hijos se obtienen a través de una consulta a base de datos.
- **LISTA**: Los hijos son fijos, se establecen mediante un array de valores.

Con estos dos modos de definición de nodos podemos ir dando la estructura que queramos al árbol.

Como en el resto de pantallas, el programador tendrá que completar un fichero TPL, un fichero PHP de views y, al menos dos, ficheros PHP de actions. Vamos a ir explicando la estructura y como trabajar en cada uno de estos ficheros.

4.1.2.1.1. Fichero de Actions

Lo primero que se debe crear es un fichero PHP donde se va a definir toda la estructura del árbol. Este fichero extenderá de la clase **gvHidraTreePatern**, esta clase nos proporcionará los métodos adecuados para crear el árbol.

El comienzo de la clase es igual que cuando trabajamos con cualquier otro tipo de panel, cargando la conexión, tablas con las que trabajar... Pasamos a crear el árbol en sí, para ello creamos un objeto de la clase **IgepArbol** a partir del que definiremos la estructura con los métodos:

- **addNodoRaiz($tipoNodo, $etiqueta, $tipoDespliegue, $datosDespliegue, $conexion)**

  Con este método crearemos el nodo raíz, hay que tener en cuenta que podemos tener más de un nodo raíz. Los parámetros que se necesitan para definirlo son:

  - *$tipoNodo*: Nombre con el que se identificará el nodo.
  - *$etiqueta*: Texto que aparecerá en pantalla asociado a este nodo. Importante no introducir retornos de carro.
  - *$tipoDespliegue*: Nos indica el modo de obtención de los nodos hijos, puede tomar dos valores "SELECT", nodos hijos vienen de BD, o "LISTA", nodos hijos fijados por un array.
  - *$datosDespliegue*: Dependiendo de lo dicho en $tipoDespliegue.
    - Si tenemos "*SELECT*": nos encontraremos un array en la que el primer parámetro se corresponderá con el identificador de los hijos de esta raiz, es decir, el primer nivel del árbol. El segundo parámetro será la select que obtendrá los valores correspondientes a este primer nivel. En la consulta SELECT, debemos definir un campo de respuesta con nombre "etiqueta" que corresponderá con el texto que aparecerá en pantalla.

```
$arbol->addNodoRaiz("ANYOS",
                    "A&ntilde;os de Facturas",
                    "SELECT",
                    array(
                         "ANYO",
                         "select anyo||' ('||count (1)||')' as \"etiqueta\",
 anyo from tinv_entradas group by anyo")
                    );
```

    - Si tenemos "*LISTA*": Tendremos un array asociativo en el que se irá dejando como índice la etiqueta de los nodos y como valor el tipo de hijo correspondiente.

```
$arbol->addNodoRaiz('DECADAS',
                    'A&ntilde;os',
                    'LISTA',
                    array(
                       'D&eacute;cada de los 70'=>'1970',
                       'D&eacute;cada de los 80'=>'1980',
                       'D&eacute;cada de los 90'=>'1990')
                   );
```

  - *$conexion*: Se puede indicar una conexión alternativa a la propia del panel para el despliegue de dicha raíz.

- **addNodoRama($tipo,$modoDespliegue,$despliegue,$dsnAlternativo = '')**

  La diferencia entre éste y el anterior es pequeña. Básicamente, tiene dos diferencias. La primera es que no tiene el parámetro *$etiqueta*, ya que la etiqueta sale de *$datosDespliegue* de su nodo padre.

  La segunda es que dentro de la definición de $datosDespliegue tiene un parámetro más destinado al paso de valores desde los nodos padre. Se trata de un array que hace referencia a los campos de cualquiera

de los padres. Estos campos son "creados" por el framework y se pueden utilizar en la creación de la select o en las etiquetas haciendo uso del caracter especial % (ver ejemplo a continuación parámetro anyo en *DGS*).

Todos los nodos del árbol excepto los que estén en la raíz se definen con este método.

- **setNodoPanel($tipoNodo, $claseManejadora, $dependencia, $conexion)**

  Este método permite al programador indicar que cierto tipo de nodo tiene asociada una representación en un panel asociado al arbol. Los parámetros que se necesitan en este método son:

  - *$tipoNodo*: Indicamos el tipo de nodo que se va a representar, que debe coincidir con alguno de los definidos con el método *addNodoRama()*.
  - *$claseManejadora*: Nombre de la clase manejadora que define ese panel.
  - *$dependencia*: Un array con los campos que debe coger de los nodos superiores para pasárselos al panel antes de realizar la búsqueda.
  - *$conexion*: Posibilidad de establecer una conexión alternativa para este panel en concreto.

A continuación vemos un ejemplo de como se crea la estructura de un árbol:

```
<?php

class PruebaArbol extends gvHidraTreePattern
{

  public function __construct()
  {
      $conf = ConfigFramework::getConfig();
      $g_dsn = $conf->getDSN('g_dsn');
      $nombreTablas= array("tinv_entradas");
      parent::__construct($g_dsn,$nombreTablas);

      $arbol = new IgepArbol();
      $arbol->addNodoRaiz("ANYOS","A&ntilde;os de
 Facturas","SELECT",array("ANYO","select anyo||' ('||count (1)||')' as
 \"etiqueta\",anyo from tinv_entradas group by anyo"));

      $arbol->addNodoRama("ANYO","LISTA",array("Por DG" =>"DGS","Por
 Trimestre"=>"TRIMESTRES","Ver Todas"=>"TODAS"));
      $arbol->addNodoRama("DGS","SELECT",array("FACTURA-DG","SELECT
 anyo,cdg,cdg as \"etiqueta\" FROM TINV_ENTRADAS WHERE anyo = '%anyo%' group
 by anyo,cdg order by cdg",array("anyo")));
      $arbol->addNodoRama("FACTURA-DG","SELECT",array("FACTURA","SELECT
 anyo||'-'||nfactura as \"etiqueta\",anyo,nfactura FROM TINV_ENTRADAS WHERE
 anyo = '%anyo%' and cdg ='%cdg%'",array("anyo","cdg")));
      ...
      $arbol->setNodoPanel("ANYO","TinvFacturasArbol",array("anyo"),"Facturas
 de %anyo%");
      $arbol->setNodoPanel("FACTURA-
DG","TinvFacturasArbol",array("anyo","cdg"),"Facturas de %anyo% de la %cdg%");
      ...
      $this->addArbol($arbol);
```

```
      ...
}
?>
```

Además de este fichero que crea la estructura principal, se debe crear un fichero php en el directorio actions por cada nodo panel que hayamos definido. Estas clases ya son clases manejadoras de un panel convencional (p.ej. un tabular, registro...).

4.1.2.1.2. Fichero de Views

A continuación debemos crear un archivo php en el directorio **views** que controle la presentación. En él se definen las asignaciones de los datos a la plantilla (tpl). En este caso no es un comportamiento genérico por lo tanto hay que instanciar la clase **IgepPanelArbol** (línea 3) que define el comportamiento del árbol, al que se le pasan los mismos parámetros que una pantalla genérica, el primero será el nombre de la *clase manejadora* que corresponde a ese panel, y el segundo, *smty_datosPanel*, variable smarty que se habrá definido en la tpl correspondiente al árbol. Activaremos la pestaña, método **activarModo**. Finalizamos agregando el árbol al panel con **agregarPanelArbol()**.

```
1 <?php
   $comportamientoVentana= new IgepPantalla();
   $panel = new IgepPanelArbol('PruebaArbol',"smty_datosPanel");
   $panel->activarModo("edi","estado_edi");
5  $comportamientoVentana->agregarPanelArbol($panel);
   $s->display('p_pruebaArbol.tpl');
  ?>
```

4.1.2.1.3. Fichero de Plantillas (TPL)

Este fichero debe contener por un lado el árbol, y por otra parte, tantos paneles como tipos de nodos representables existan en el panel, los indicados con el método **setNodoPanel()**. Es importante tener en cuenta que se ha de mostrar sólo el panel activo, y para ello utilizamos la clausula *{if}* de Smarty, en la variable *$smty_panelVisible* nos vendrá el panel que queremos activo, es decir, nos vendrá el identificador que le habremos dado cuando hemos definido los nodos.

A continuación mostramos la tpl utilizada para el ejemplo:

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=
$smty_descBreve textoAviso=$smty_textoAviso}
{CWBarra usuario=$smty_usuario codigo=$smty_codigo}
    {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
{/CWBarra}
{CWMarcoPanel conPestanyas="true"}
<!-- ****************** PANEL edi ***********************-->
{CWArbol estado=$estado_edi arbol=$smty_objArbol}
    {if $smty_panelVisible == "ANYOS" || $smty_panelVisible == "FACTURA-DG"}
        {CWPanel id="edi" tipoComprobacion="envio" itemSeleccionado=
$smty_filaSeleccionada action="operarTabla" method="post"
 estado="on" claseManejadora="TinvFacturasArbol" accion=
$smty_operacionFichaTinvFacturasArbol}
        {CWBarraSupPanel titulo=$smty_tituloPanel}
            {CWBotonTooltip imagen="01" titulo="Insertar registro"
 funcion="insertar" actuaSobre="ficha"}
            {CWBotonTooltip imagen="02" titulo="Modificar registro"
 funcion="modificar" actuaSobre="ficha"}
```

```
            {CWBotonTooltip imagen="03" titulo="Eliminar registro"
 funcion="eliminar" actuaSobre="ficha"}
            {CWBotonTooltip imagen="04" titulo="Limpiar registro"
 funcion="limpiar" actuaSobre="edicion"}
        {/CWBarraSupPanel}
        {CWContenedor}
            {CWFichaEdicion id="FichaEdicion" datos=$smty_datosPanel
 numPagInsertar="1"}
                {CWFicha}
                        {CWCampoTexto nombre="anyo" editable="true" size="4"
 value=$smty_anyoNuevo textoAsociado="A&ntilde;o"}
                          
                        {CWCampoTexto nombre="nfactura" editable="false"
 size="6" value ="0" textoAsociado="N&uacute;mero Factura"}
                        <br /><br />
                        {CWLista nombre="procedencia"  radio="true"
 editable="true" datos=$smty_datosPreInsertadosTinvFacturasArbol.procedencia
 textoAsociado="Procedencia"}
                        <br /><br />
                        {CWCampoTexto nombre="cif" editable="true" size="13"
 maxlength="13" textoAsociado="Cif"}
                         
                        {CWCampoTexto nombre="orden" editable="true" size="2"
 maxlength="2" textoAsociado="Orden" actualizaA="nombre"}
                         {CWCampoTexto nombre="nombre" editable="false"
 size="40"}
                        {CWBotonTooltip imagen="13" titulo="Ventana de
 seleccion" funcion="abrirVS" actuaSobre="cif"}
                        <br /><br />
                        {CWLista nombre="cdg" size="2" editable="true" datos=
$smty_datosPreInsertadosTinvFacturasArbol.cdg textoAsociado="D.General"}
                        {CWCampoTexto nombre="fadquisicion" editable="true"
 size="10" conCalendario="true" textoAsociado="Fecha Adq"}
                        <br /><br />
                        {CWCampoTexto nombre="fcertificacion" editable="false"
 size="10" textoAsociado="Fecha Certif"}
                          
                        {CWCampoTexto nombre="codFactura" editable="true"
 size="13" maxlength="13" textoAsociado="Cod. Factura"}
                        </br></br>
                {/CWFicha}
                {CWPaginador enlacesVisibles="3"}
            {/CWFichaEdicion}
        {/CWContenedor}
        {CWBarraInfPanel}
            {CWBoton imagen="41" texto="Guardar" class="boton"
 accion="guardar" vCamposComprobar=$smty_validacionesTinvFacturasArbol}
            {CWBoton imagen="42" texto=" Cancelar" class="boton"
 accion="guardar" action="IgepPruebaArbol__cancelarArbol"}
        {/CWBarraInfPanel}
    {/CWPanel}
  {else}
```

```
    {CWPanel id="edi" tipoComprobacion="envio" action="operarTabla"
 method="post" detalleDe="edi" estado=$estado_edi claseManejadora=""}
        {CWBarraSupPanel titulo=""}
        {/CWBarraSupPanel}
        {CWContenedor}
            {CWFichaEdicion  id="FichaDetalle" datos=$smty_datosPanel
 numPagInsertar="0"}
                {CWFicha}
                   </br></br></br></br>
                   NO HAY DATOS
                   </br></br></br></br>
                {/CWFicha}
            {/CWFichaEdicion}
        {/CWContenedor}
        {CWBarraInfPanel}
            {CWBoton imagen="42" texto=" Cancelar" class="boton"
 accion="guardar" action="IgepPruebaArbol__cancelarArbol"}
        {/CWBarraInfPanel}
    {/CWPanel}
  {/if}
{/CWArbol}
<!-- ****************** PESTAÑAS ************************-->
    {CWContenedorPestanyas}
        {CWPestanya tipo="edi" estado=$estado_edi}
    {/CWContenedorPestanyas}
{/CWMarcoPanel}
{/CWVentana}
```

4.1.2.2. Otro ejemplo

A continuación se muestra un ejemplo en el que la definición de jerarquía está definida en un único campo (subconcepto) y en funcion de su longitud obtenemos el tipo de nodo. La jerarquía de nodos buscada es:

```
ANYOS
   ANYO
      CAP1    (longitud 1)
         CAP2    (longitud 2)
            CAP3    (longitud 3)
               CAP5   (longitud 5)
```

La forma de definir los nodos seria, partiendo de cada nodo raíz, ir definiendo como se obtienen los nodos hijos y así sucesivamente hasta tener definidos todos los nodos. Puesto que el nodo CAP5 no tiene hijos no necesitamos definir ese nodo.

```
...
$arbol->addNodoRaiz('ANYOS', 'A&ntilde;os de Subconceptos', 'SELECT',
        array('ANYO',"select concat(concat(concat(anyo,' ('),count(1)),')') as
 \"etiqueta\", anyo
                      from tcom_subconceptos_anyo group by anyo"));
$arbol->addNodoRama('ANYO',  'SELECT',
        array('CAP1',"SELECT anyo, subconcepto, concat(concat(subconcepto,'
 '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo WHERE anyo = %anyo% and
 length(subconcepto)=1
```

```
                      order by subconcepto", array("anyo")));
$arbol->addNodoRama('CAP1',  'SELECT',
        array('CAP2',"SELECT anyo, subconcepto, concat(concat(subconcepto,'
 '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo
                      WHERE anyo = %anyo% and
 substr(subconcepto,1,1)='%subconcepto%' and length(subconcepto)=2
                      order by anyo,subconcepto",
 array("anyo",'subconcepto')));
$arbol->addNodoRama('CAP2',  'SELECT',
        array('CAP3',"SELECT anyo, subconcepto, concat(concat(subconcepto,'
 '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo
                      WHERE anyo = %anyo% and
 substr(subconcepto,1,2)='%subconcepto%' and length(subconcepto)=3
                      order by anyo,subconcepto",
 array("anyo",'subconcepto')));
$arbol->addNodoRama('CAP3',  'SELECT',
        array('CAP5',"SELECT anyo, subconcepto, concat(concat(subconcepto,'
 '),descrip) as \"etiqueta\"
                      FROM tcom_subconceptos_anyo
                      WHERE anyo = %anyo% and
 substr(subconcepto,1,3)='%subconcepto%' and length(subconcepto)=5
                      order by anyo,subconcepto",
 array("anyo",'subconcepto')));
...
```

4.2. Componentes complejos

En este punto vamos a hablar de dos componentes que se salen de los básicos para diseñar una pantalla genérica y que nos ayudarán a la hora de trabajar con grupos de datos, tanto para seleccionar como para agrupar.

4.2.1. Ventana de selección

Las ventanas de selección es una particularización de las listas, siendo las siguientes características las que las diferencian:

- Se pueden filtrar los valores posibles mediante unas búsquedas sencillas. Lo que las hace aconsejables cuando el número de elementos posibles es superior a 100 aprox.
- Se puede visualizar más información complementaria que nos ayude a la selección de un registro, es decir, en la ventana de selección se mostrarán tantas columnas como se necesiten para la mejor comprensión del registro.
- Del registro seleccionado podemos recuperar otra información que nos interese para llevárnosla a otros campos del panel origen. Esta información puede ser visible o no en la ventana de selección.
- Nos permite tener diferentes fuentes de datos, los datos pueden venir de un web service, de base de datos, de una clase...
- Se puede crear una ventana de selección de imágenes.

Al igual que las listas, la definición de una ventana de selección se definen en dos pasos. Por un lado, se debe definir la ventana de selección en el AppMainWindow (como las listas). Por otro, se debe incorporar en la clase manejadora

4.2.1.1. Definición de la ventana

Para definir una ventana de selección tenemos que definirla en la clase manejadora. Creamos una instancia de la clase **gvHidraSelectionWindow()** pasándole el nombre de campo de la tpl, campo del que dependerá la ventana de selección, y como segundo parámetro, un identificador (p.ej. USUARIOS) del origen de los datos, opcionalmente, podemos indicar una conexión alternativa como tercer parámetro, para ello debemos indicar pasar un DSN de conexión. Después tendremos que ir relacionando los campos del panel que van a ser actualizados con el campo correspondiente de la base de datos, con el método **addMatching()**, como mínimo habrá una relación ya que de lo contrario no actualizaríamos ningún campo de la ventana orígen. Por último añadimos esta definición de la ventana al panel con **addSelectionWindow()**.

```
$usuarios = new gvHidraSelectionWindow('filUsuario','USUARIOS');
$usuarios->addMatching('filId','id');
$usuarios->addMatching('filUsuario','usuario');
$usuarios->addMatching('filNombre','nombre');
$this->addSelectionWindow($usuarios);
```

Además de esta definición básica la ventana tiene dos métodos que nos permitirán customizarla:

- **setSize()**: este método permite fijar el tamaño de la ventana emergente.

```
$usuarios = new gvHidraSelectionWindow('usuario','USUARIOS');
...
$usuarios->setSize(800,600);
...
```

- **setLimit()**: este método fija el número máximo de elementos que mostrará. Si el resultado de la consulta excede este número, mostrará un mensaje advirtiendo al usuario de que no se mostrarán todos los resultados obtenidos.

```
$usuarios = new gvHidraSelectionWindow('usuario','USUARIOS');
...
$usuarios->setLimit(10);
...
```

- **setRowsNumber()**: este método fija el número de filas que se mostrarán por pantalla simultaneamente.

```
$usuarios = new gvHidraSelectionWindow('usuario','USUARIOS');
...
$usuarios->setRowsNumber(4);
...
```

- **setTemplate()**: nos permite diseñar nuestra propio contenido de la ventana de selección, es decir, tendrá una tpl particular en la que podremos seleccionar el tipo de componente (CampoTexto, List, Imagen,), el nombre de la columna, el ancho de las mismas, ...

```
$usuarios = new gvHidraSelectionWindow('usuario','USUARIOS');
...
$usuarios->setTemplate('ventanaSeleccion/incUsuarios.tpl');
...
```

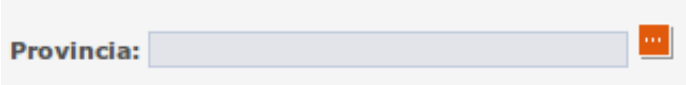
Si utilizamos este método significa que tenemos una plantilla (tpl) que dará formato a la ventana de selección. En caso contrario, el framework dibuja una ventana de selección sencilla, en la que su contenido no es configurable.

Siguiendo el caso del ejemplo, debemos tener un directorio dentro del directorio plantillas, llamado "ventanaSeleccion", y dentro tendremos las tpl que correspondan a las ventanas de selección de la aplicación. En esa plantilla se crearán los campos que queremos que nos sean necesarios, tanto visibles como ocultos.

```
{CWImagen nombre="fichero" rutaAbs="yes" textoAsociado="Fichero" width="40"
 height="50" bumpbox="true"}
{CWCampoTexto nombre="nombre" size="15" textoAsociado="Nombre"}
{CWCampoTexto nombre="descripcion" size="40" textoAsociado="Descripción"}
{CWCampoTexto nombre="id" oculto="true"}
```

Una vez definida la ventana sólo nos queda introducir en la tpl, que invocará la ventana de selección, un componente CWBotonTooltip que haga referencia a ella. Para ello se parametriza indicando el campo sobre el que actua (en nuestro ejemplo usuario), el form y el panel. Aqui tenemos un ejemplo:

```
{CWCampoTexto nombre="filUsuarios" size="8" textoAsociado="Usuario"}
{CWBotonTooltip imagen="13" titulo="Busqueda de Usuarios"
 funcion="abrirVS" actuaSobre="filUsuarios"}
```



Al pulsar en el botón tooltip de la ventana nos aparecerá una ventana emergente como la siguiente:

4.2.1.2. Fuentes de datos

Las definiciones particulares a la aplicación, se realizarán, al igual que en el caso de las listas, se definirán en el arranque de la aplicación. Para ello debemos de añadir su definición en la clase que controle el panel principal de la aplicación, fichero **AppMainWindow.php**. Ya que el origen de los datos puede ser diferente, tenemos dos formas de indicarlo:

- **setSelectionWindow_DBSource($key, $query, $fields=null)**:

  Método para introducir fuente de datos con consulta SQL.

  El primer parámetro será el identificador que nos es necesario cuando definimos la ventana en la clase manejadora. El segundo parámetro es la propia consulta que nos devolverá los datos a mostrar. El tercer parámetro, opcional, nos permite añadir campos a los que ya aparecen en la consulta, sobre ellos también se aplicará el filtro de búsqueda.

  En la consulta también podemos usar subconsultas. En el caso de postgres, estas han de tener siempre alias en el from (aunque no se vaya a usar) por lo que conviene ponerlo siempre (para cualquier tipo de base de datos). Notar también que en la subconsulta sólo pondremos alias para los campos que queramos renombrar, no afectan las mayúsculas / minúsculas.

```
$conf->setSelectionWindow_DBSource('USUARIOS','select id, usuario, nombre from
 usuarios');
```

- **setSelectionWindow_ClassSource($key, $query)**:

  Método para introducir fuente de datos con una clase. Esta clase debe implementar la interfaz **gvHidraSelectionWindow_Source.php**.

```
$conf->setSelectionWindow_ClassSource('PERSONAS',"PersonasSource");
```

4.2.1.3. Búsqueda en la ventana

La búsqueda, por defecto, busca el texto introducido en cualquiera de los campos que se visualizan, y si obtenemos los datos de una consulta a bd también buscará en el array de campos añadidos. Podemos cambiar este comportamiento de forma similar a los filtros en paneles. Veamos un ejemplo:

```
$usuarios = new gvHidraSelectionWindow('filUsuario','USUARIOS');
$usuarios->addMatching('filId','id');
$usuarios->addMatching('filUsuario','usuario');
$usuarios->addMatching('filNombre','nombre');
$usuarios->setQueryMode(2); // valor por defecto es 1
$this->addSelectionWindow($usuarios);
```

Por defecto, cuando el usuario introduce un texto para buscar, busca por todos los campos que aparecen en la consulta, concatenados y separados por un espacio: concat(concat(col1,' '),col2)... De esta forma el usuario puede hacer una búsqueda que coincida con el final de un campo y el principio del siguiente. Es importante que, para las dos fuentes de datos, los parámetros del matching son los que recogerán los valores y los introducirán en la ventana destino.

4.2.1.4. Dependencia

Al igual que en las listas, tenemos la posibilidad de crear ventanas de selección dependientes de otros controles, aunque, en este caso, podemos definir dos tipos de dependencia (fuerte o débil). La fuerte añade a la WHERE de la consulta que se genere en la ventana una clausula con el valor del campo del que

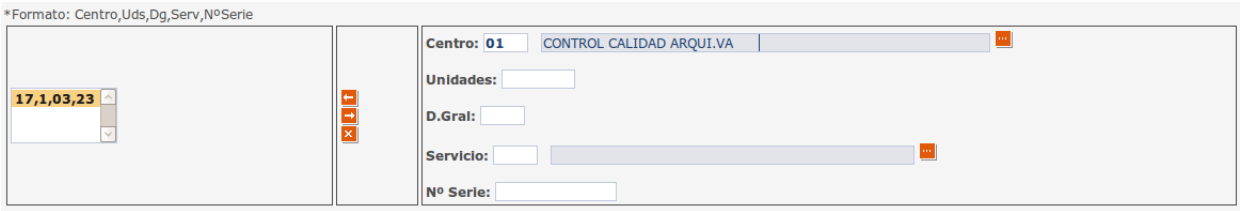
depende sin tener en cuenta si este tiene valor o no. La débil, sólo lo hace si el campo tiene valor. Para indicar la dependencia tenemos que hacer uso del método setDependence indicando los campos de la tpl, su correspondencia en la BD y el tipo de dependencia deseado :0-> fuerte(valor por defecto) o 1-> débil. En nuestro ejemplo, sería:

```
$usuarios = new gvHidraSelectionWindow('filUsuario','USUARIOS');
$usuarios->addMatching('filId','id');
$usuarios->addMatching('filUsuario','usuario');
$usuarios->addMatching('filNombre','nombre');
$usuarios-
>setDependence(array('esDePersonal'),array('tper_ocupacion.esPersonal'),0);
$this->addSelectionWindow($usuarios);
```

4.2.2. Selector

El selector corresponde con el plugin CWSelector, es un plugin que aún está poco elaborado, se creó para representar relaciones 1:N dentro de un mismo panel. De todos modos, la implementación es manual, es decir. el programador será el que realice las operaciones.

Aquí vemos el aspecto que nos dará el selector en una pantalla.



Vamos a explicar como hacer uso de él:

- *En la plantilla (tpl)*:

  El plugin CWSelector solamente aparecerá dentro de una ficha (CWFicha). CWSelector es un plugin block, dentro del cual podremos definir otros componentes, como CWCampoTexto y CWLista.

```
{CWSelector titulo="TituloDelGrupo" botones=$smty_botones nombre="listaBienes"
 editable="true" separador="."}
    {CWCampoTexto nombre="CampoTexto1" editable="false" size="6"
 textoAsociado="CampoTexto1"}
    {CWCampoTexto nombre="CampoTexto2" editable="false" size="6"
 textoAsociado="CampoTexto1"}
    {CWCampoTexto nombre="CampoTexto3" editable="false" size="6"
 textoAsociado="CampoTexto1"}
{/CWSelector}
```

Los valores introducidos en estos campos serán copiados, cuando pulsamos el botón , a la lista múltiple que tenemos en la parte izquierda, separando los valores por el caracter introducido en el parámetro *separador*, en nuestro ejemplo el punto.

Con el botón , copiaremos los valores que estén seleccionados en la lista múltiple a los campos que hayamos definido dentro del selector.

Finalmente, el botón nos servirá para eliminar el registro que tengamos seleccionado en la lista múltiple.

- *En el views*:

  En el views tenemos que indicar qué botones del selector tendremos activos, para ello asignamos la variable smarty *smty_botones*.

```
$botones = array('insertar','modificar','eliminar');
$s->assign("smty_botones",$botones);
```

- *En la clase manejadora*:

  Finalmente, ¿como recuperamos el contenido del Selector en la clase manejadora? Recordemos que el contenido del selector devolverá una lista, ya que podemos tener N items, pero al recibirlo en la clase manejadora, sólo recibimos un string de elementos separados por separador. Por esta razón, para obtener un array podemos utilizar el siguiente código:

```
public function postInsertar($objDatos) {

    $m_datos = $objDatos->getAllTuplas();
    foreach($m_datos as $tupla) {
       if($tupla['listaBienes']!='') {
           echo 'El resultado es: ';
           print_r($tupla['listaBienes']);die;
       }
    ...
}
```

Si hubiéramos introducido los siguientes valores en nuestro ejemplo:

El resultado sería:

```
Array
(
    [0] => 1.2.3
    [1] => 4.5.6
)
```

4.3. Tratamiento de ficheros

Vamos a contar como trabajar con documentos y ficheros de cualquier tipo. Añadimos ejemplos de como insertar un fichero en la BD, como mostrar una imagen que se encuentra en el servidor en un campo CWImagen, como insertar N tuplas en una BD a partir de un fichero XML.

4.3.1. Manejo de ficheros de imágenes

En este primer ejemplo vamos a mostrar como crear un mantenimiento que almacene imágenes en la BD (en nuestro ejemplo PostgreSQL) y las muestre en un **CWImagen**. Para conseguir esto necesitamos que en la interfaz (en la tpl) aparezcan dos plugins especiales de tratamiento de ficheros (**CWUpLoad** y el **CWImagen**).

El primero de ellos, el **CWUpLoad**, nos permite seleccionar un fichero ubicado en el ordenador cliente y subirlo al servidor Web. El contenido de este fichero se almacena en una ubicación temporal y el progra-

mador es el encargado de hacer uso de él. Posteriormente mostraremos los métodos que gvHidra proporciona para que se tenga acceso a la información de dicho fichero.

El segundo de ellos es el **CWImagen**, que permite mostrar una imagen en un panel a partir de una URL.

A continuación mostramos un ejemplo del panel resultado:

Para conseguir esto tenemos que tener en cuenta las dos partes que van a suponer trabajo para el programador, por un lado el tratamiento del fichero imagen cuando el usuario lo selecciona con el **CWUpLoad** y lo quiere almacenar en el servidor y, por otro, la representación de un fichero imagen en el **CWImagen**. El primero de estos pasos necesita de un acceso a la información del fichero y la inserción del mismo en la tupla que se esté modificando. A continuación mostramos el código que se ha insertado en el panel para resolverlo.

```
public function postModificar($objDatos)
{
    do
    {
      $datosFile = $objDatos->getFileInfo('ficheroUpload');
      if(is_array($datosFile) and ($datosFile['tmp_name']!='')){
        $fichero = pg_escape_bytea(file_get_contents($datosFile['tmp_name']));
        $res = $this->operar("update tinv_imagenes SET fichero='{$fichero}',
 nombre='".$datosFile['name']."' where id=".$objDatos->getValue('id'));
        if($res==-1)
        {
          $this->showMessage('APL-25');
          return -1;
        }
      }
    } while($objDatos->nextTupla());
    return 0;
  }
```

De este código destaca, en primer lugar, el método **getFileInfo()** del objeto objDatos. Este método devuelve el array de información del HTTP que indica los datos del fichero que ha subido el usuario (nombre, tipo, ubicación,...). A partir de esta información el programador ya puede hacer uso del fichero; y, en este caso, decide insertarlo en una BD PostgreSQL. Para ello va a hacer uso de un campo de tipo Bytea

y, por esta razón, utiliza las siguientes premisas. Primero, tras obtener el contenido del fichero (función php *file_get_contents*) utiliza la función nativa de php *pg_escape_bytea* para poder construir la consulta. Construye la consulta incluyendo el contenido (escapado) del fichero entre llaves. De este modo la instrucción no tiene problemas al pasar el array de bytes al PosgreSQL. El otro punto complejo de esta pantalla está en la recuperación de la imagen de la BD. Para ello debemos obtener la imagen del campo bytea y almacenarla en una ubicación accesible por nuestro plugin **CWImagen**. Por esta razón, en este caso hemos decidido crear un directorio imagenes en la raiz de nuestro proyecto (también se podria usar el templates_c que ya usamos para smarty) donde se crearán los ficheros que se quieran visualizar. Luego se asignará al **CWImagen** la ruta de cada uno de ellos. El código que se encarga de realizar esto es:

```
public function postEditar($objDatos)
{
    //Comprobamos si tiene imagen y en este caso la cargamos.
    $res = $objDatos->getAllTuplas();
    foreach($res as $indice => $tupla)
    {
      if($tupla['fichero']!='')
      {
        $nuevoFichero = "imagenes/".$tupla['nombre'];
        if(!file_exists($nuevoFichero))
        {
          $imagen = pg_unescape_bytea($tupla['fichero']);
          $gestor = fopen($nuevoFichero, 'x');
          fwrite($gestor,$imagen);
        }
        $res[$indice]['fichero'] = $nuevoFichero;
      }
    }
    $objDatos->setAllTuplas($res);
}//Function postEditar
```

Como se puede suponer del código anterior, el CWImagen se llama fichero.

```
{CWImagen nombre="fichero"}
{CWUpLoad nombre="ficheroUpload" size="10" textoAsociado="Si quieres cambiar
 la imagen..."}
```

4.3.2. Manejo de ficheros de cualquier tipo

Puede darse el caso que queramos almacenar documentos de cualquier tipo en la BD. Para ello tendríamos que hacer una simple modificación al código anterior. Es evidente que ya no nocesitariamos el plugin CWImagen. Lo único que necesitamos es un botón que lance una acción particular que recupera el documento de la BD y lance el mismo a una nueva ventana del navegador. El código que se encarga de realizar esto es el siguiente

```
public function accionesParticulares($str_accion, $objDatos)
{
    switch ($str_accion)
    {
      case 'verFichero':
        //Bucle para crear un listado para cada peticion seleccionada:
        $objDatos->setOperation("seleccionar");
        $m_datosSeleccionados = $objDatos->getAllTuplas();
```

```
        $res = $this->consultar("SELECT fichero,tipo from tinv_documentos
 WHERE id = ".$m_datosSeleccionados[0]['id']);
        if($res[0]['fichero']!='')
        {
          header('Content-Type: '.$res[0]['tipo']);
          header('Content-Disposition: attachment; filename="'.
$m_datosSeleccionados[0]['nombre'].'"');
          print(pg_unescape_bytea($res[0]['fichero']));
          ob_end_flush ();
          //Para que no continue la ejecución de la página
          die;
        }
        $actionForward = $objDatos->getForward('correcto');
       break;
    }
    return $actionForward;
}
```

Como nota podemos indicar que, a diferencia del ejemplo anterior, en este ejemplo, al almacenar el documento en la BD se almacena no sólo su contenido, sino también el tipo. De este modo se puede asignar al header para que el navegador lo abra con el programa adecuado para tratarlo dependiendo de la extensión (MIME-TYPES).

4.3.3. Importar datos a la BD desde fichero

Este es otro de los ejemplos típicos con los que nos podemos encontrar. Imaginemos el caso de una aplicación que carga sus datos a partir de una fuente externa como, por ejemplo, un XML. Para este ejemplo sólo utilizaremos el plugin CWUpLoad y el código sería el siguiente:

```
public function accionesParticulares($str_accion, $objDatos)
{
    switch ($str_accion)
    {
      case 'importarFichero':
        $objDatos->setOperation('external');
        $datosFile = $objDatos->getFileInfo('ficheroUpload');
        $contenidoFichero =
 utf8_encode(file_get_contents($datosFile['tmp_name']));
        $dom = domxml_open_mem($contenidoFichero,
          DOMXML_LOAD_PARSING + //0
          DOMXML_LOAD_COMPLETE_ATTRS + //8
          DOMXML_LOAD_SUBSTITUTE_ENTITIES + //4
          DOMXML_LOAD_DONT_KEEP_BLANKS //16
        );
        if(is_object($dom))
        {
          $xpathXML = $dom->xpath_new_context();
          $xpresultXML = $xpathXML->xpath_eval("//persona", $xpathXML);
          $v_personas = $xpresultXML->nodeset;
          //preparamos los contadores
          $insertados = 0;
          $totales = 0;
          foreach($v_personas as $persona)
```

```
          {
            $nif = $persona->get_attribute('nif');
            $comprobacion = $this->consultar('SELECT count(1) as "cuenta" FROM
 tinv_xml where nif=\''.$nif.'\'');
            if($comprobacion[0]["cuenta"]==0)
            {
              $nombre = $persona->get_attribute('nombre');
              $apellidos = $persona->get_attribute('apellidos');
              $correcto = $this->operar("INSERT INTO tinv_xml values ('".
$nif."','".$nombre."','".$apellidos."')");
              if($correcto!=-1)
                $insertados++;
            }
            $totales++;
          }
          $this->showMessage('APL-26',array($insertados,$totales));
        }
        else {
          $this->showMessage('APL-27');
        }
        $this->setResultForSearch(array());
        $actionForward = $objDatos->getForward('correcto');
        break;
      default:
        die("La acción $str_accion no está activada para esta clase.");
    }
    return $actionForward;
  }
```

Como podemos ver, la aplicación espera un fichero XML del que extrae todos los tags <persona>. Si no existen en la BD los inserta con los atributos deseados.

4.4. Control de la Navegación. Saltando entre ventanas

Uno de los típicos problemas en el diseño de aplicaciones web es el control de la navegación. Para ayudar en la resolución de este problema, el framework incorpora un mecanismo que facilita el uso de estos saltos entre páginas con el consiguiente paso de mensajes.

Para entender la problemática que implica los saltos entre páginas, vamos a destacar ciertos puntos que se deben considerar:

- **Paso de parámetros**: evidentemente, se debe poder pasar información desde el origen hasta el destino. En nuestro caso desde una clase manejadora a otra. Para ello, tenemos que hacer uso de la SESSION.
- **Pila de saltos**: de alguna manera, una vez volvamos de un salto debemos limpiar todo rastro del salto, para evitar colisiones con futuras operaciones.
- **Automatización del proceso**: intentar que el proceso sea lo más limpio posible para el programador.
- **Momento del salto/modo de regreso**: dependerá de las necesidades, necesitaremos saltar y, tras la vuelta, recargar la ventana o, por el contrario, mantener la información. Por ejemplo, si queremos saltar

en mitad de un proceso de inserción de datos del usuario, necesitaremos regresar a la ventana origen sin perder la información disponible.

Teniendo en cuenta estas consideraciones, el framework ha implementado una clase y una serie de métodos con la intención de facilitar el proceso al programador.

Nota: Hay que destacar que un salto puede ser tan simple como indicar una URL en el actionForward correspondiente; pero con ello perderiamos todo control sobre el retorno (no podríamos garantizar la "limpieza" de la SESSION).

Para ilustrar los saltos, hemos distinguido dos tipos o patrones de saltos que en los siguientes puntos se desarrollaran en dos ejemplos:

- **Acumulador/Operador**: se trata de los saltos en los cuales debemos saltar a una clase para seleccionar/operar con un conjunto de datos y volver a la clase origen.
- **Clave Ajena**: se trata de los saltos en los que, al mantener una clase, necesitamos modificar datos de una tabla que tiene relación directa con él (insertar un nuevo elemento). Un ejemplo clásico sería el mantenimiento de facturas cuando damos de alta una factura y el proveedor no existe. En estos saltos, es vital mantener los datos de origen una vez se salta al regresar (el usuario estaba a mitad de edición). La forma de resolverlos será con el uso de saltos modales.

4.4.1. Acumulador/Operador

Vamos a implementar el ejemplo de un salto con recarga en el retorno. Tenemos un mantenimiento tabular en el que vamos a acumular fila a fila una serie de aspirantes a los que les queremos realizar una entrevista grupal. Estos aspirantes se obtienen a través de otra ventana (también tabular) en la que podemos buscar los aspirantes por sus perfiles y seleccionarlos para la entrevista. Una vez seleccionados los aspirantes deseados, se pulsa al botón "Citar para entrevista" disparando el proceso (envío de correos, inserción en el planning del entrevistador, ...).

Ficheros implicados:

Tabla 4.1. Ficheros implicados en implementación

clase manejadora	view	plantilla tpl	descripción
GeneradorEntrevistas.php	p_generadorEntrevistas.php	p_generadorEntrevistas.tpl	Generador de entrevistas. Origen del salto
Aspirantes.php	p_aspirantes.php	p_aspirantes.tpl	Mantenimiento de aspirantes.

Dando por hecho la creación de los dos mantenimientos vamos a ir, paso a paso, indicando como implementar el salto. En primer lugar, tenemos que ubicar un punto de disparo de la acción de salto en el origen del salto (p_generadorEntrevistas.tpl). Puede ser bien un botón clásico:

```
{CWBoton id="saltoVisualizar" imagen="49" texto="SALTO" class="boton"
 accion="saltar"}
```

o un botón tooltip. Si elegimos esta opción hay que tener en cuenta dónde se va a ubicar dicho botón:

- Podemos querer que el botón actúe de forma general al registro, por lo tanto se pondrá en la barra superior del panel. A destacar que el parámetro *actuaSobre*, en este caso, debe ser "*ficha*" o "*tabla*".

```
{CWBarraSuperior}
  {CWBotonTooltip imagen="39" titulo="+" funcion="saltar" actuaSobre="ficha"}
{/CWBarraSuperior}
```

- En cambio nos puede interesar que el salto modal afecte a uno o varios campos en concreto, entonces el botón tooltip se encontraría dentro de la ficha o fila, según el caso. Aquí el parámetro actuaSobre debe contener el nombre del campo al que vaya a afectar.

```
{CWFicha}
  ...
  {CWCampoTexto nombre="ediCif" size="9" editable="true" textoAsociado="CIF"
 dataType=$smty_dataType_Personas.ediCif}
  {CWBotonTooltip imagen="39" titulo="+" funcion="saltar" actuaSobre="ediCif"}
  ...
{/CWFicha}
```

En este ejemplo nos quedamos con la primera opción.

Ahora, tenemos que crear en la clase manejadora origen del salto (GeneradorEntrevistas), un método que gestione el salto (si se debe saltar o no; si pasa parámetros). En este caso el framework proporciona un método para tal uso **saltoDeVentana**:

```
public function saltoDeVentana($objDatos, $objSalto) {

    //Nombre de la claseM destino del salto
    $objSalto->setNameTargetClass('Aspirantes');
    //Nombre de la accion de entrada a la ventana
    $objSalto->setTargetAction('iniciarVentana');
    //Nombre de la accion de retorno
    $objSalto->setReturnAction('iniciarVentana');
    //Parametro: numero maximo de seleccionables
    $objSalto->setParam('numeroMaxSeleccionable',5);
    return 0;
}
```

Como se puede ver, por un lado tenemos los datos relativos a la pantalla (objDatos) y por otro, los relativos al salto. El objeto salto es el que se deberá configurar, para indicar la clase destino del salto, **setNameTargetClass()**, con el método **getNameSourceClass()** podremos obtener el nombre de la clase origen, la acción de entrada en la nueva ventana (el mapeo correspondiente), **setTargetAction()**, también se estable la acción de retorno a la ventana origen con **setReturnAction()**. También se pueden añadir parámetros al salto usando el método **setParam()**.

Este código, redireccionará la ejecución hacia la clase *Aspirantes* entrando con la acción *iniciarVentana*. Esto implica que en el fichero de mapeos (mappings.php) tenemos que definir el _AddMapping y los _AddForwards correspondientes.

Ahora, en la clase destino del salto, debemos recoger la información transmitida en el salto. Para ello, en el constructor añadiremos el siguiente código:

```
//Comprobamos si hay salto
$objSalto = IgepSession::dameSalto();
if(is_object($objSalto)){
    $this->parametros = $objSalto->getParams();
```

```
}
```

Tenemos que acceder a la zona de la SESSION donde el framework almacena la información del salto y recuperar el objeto con **dameSalto()** y con **getParams()** obtendremos los parámetros que nos pasará la clase origen.

Una vez realizado el salto, vamos a indicar como se realiza el retorno. Para ello primero vamos a ver en la tpl como indicar que vamos a lanzar una acción de retorno. En nuestro caso hemos añadido dos botones que controlan el retorno, uno acumula los seleccionados, el otro cancela la operación. En ambos casos la acción es volver, los diferenciamos a través del id.

```
{CWBoton id="acumular" imagen="49" texto="ACUMULAR" class="boton"
 accion="volver"}
{CWBoton id="cancelarAcumular" imagen="42" texto="CANCELAR" class="boton"
 accion="volver"}
```

Este estímulo se recibirá en la clase objeto del salto (en nuestro ejemplo Aspirantes). En ella debemos sobreescribir el método regresoAVentana del framework que nos permitirá gestionar el regreso de este salto.

```
public function regresoAVentana($objDatos, $objSalto){

    if($objSalto->getId()=='acumular'){
        //Recogemos los datos seleccionados por el usuario en pantalla
        $m_datos = $objDatos->getAllTuplas('seleccionar');
        $objSalto->setParam('nuevosAspirantes',$m_datos);
        $objSalto->setReturnAction('acumular');
    }
    return 0;
}
```

En nuestro ejemplo, en el caso de que el usuario haya seleccionado aspirantes y haya pulsado acumular, volveremos a la clase con la acción acumular, podremos saber si se ha pulsado el botón Acumular obteniendo su id con el método **getId()**. En caso contrario volveremos con la acción que se programó en el salto (iniciarVentana).

Por supuesto, la clase origen del salto, tiene que tener en el fichero de mapeo el *_AddMapping* y los *_AddForwards* correspondientes (en nuestro caso GeneradorEntrevistas__acumular y GeneradorEntrevistas__iniciarVentana).

```
$this->_AddMapping('GeneradorEntrevistas__acumular', 'GeneradorEntrevistas');
$this->_AddForward('GeneradorEntrevistas__acumular', 'correcto', 'index.php?
view=views/p_generadorEntrevistas.php');
```

Nota: Hay una serie de métodos que no se han utilizado en este ejemplo pero que pueden ser de utilidad. Entre ellos destacamos el método getNameSourceClass que devuelve el nombre de la clase origen del salto.

Finalmente, en la clase origen del salto, tenemos que configurar el regreso del mismo. En el caso de un regreso por cancelación de acción, no tenemos que hacer nada, ya que se trata de una acción genérica del framework (iniciarVentana). En el caso de un regreso para acumular, se ha decidido realizar una acción no genérica: una acción particular. Se trata de la acción GeneradorEntrevistas__acumular.

Para tratarla tenemos que incorporar el siguiente código.

```
public function accionesParticulares($accion, $objDatos) {
    switch ($accion) {
        case 'acumular':
            $salto = IgepSession::damePanel('saltoIgep');
            $nuevos = $salto->getParam('nuevosAspirantes');
            $this->acumularAspirantes($nuevos);
            $actionForward = $objDatos->getForward('correcto');
            break;
        }
        return $actionForward;
}
```

En el método acumularAspirantes, se realizarán las acciones pertinentes según la lógica del caso de uso. En nuestro ejemplo tendriamos que añadir los nuevos aspirantes a los seleccionados previamente:

```
public function acumularAspirantes($nuevos)
{
   $acumulados = $this->getResultForSearch();
   if($acumulados == '')
      $acumulados = array();
   foreach($nuevos as $indice =>$linea)
   {
      $aspirante = array();
      $aspirante["dni"] = $linea["dni"];
      $aspirante["nombre"] = $linea["nombre"];
      ...
      array_push($acumulados,$aspirante);
   }
   $this->setResultForSearch($acumulados);

   // si en vez de mostrar los aspirantes seleccionados quisieramos
 modificarlos
   // directamente en la base de datos, luego podriamos refrescar la pantalla
 con:
   //$this->refreshSearch();
}
```

Como podemos observar al ejecutar, al regresar del salto, recargamos la ventana de origen. En este caso es necesario ya que hemos modificado el resultado de su búsqueda. Este ejemplo, se puede realizar con un salto clásico (como hemos visto), o con el uso de una ventana modal. En el siguiente ejemplo veremos como realizar un salto con ventana modal.

4.4.2. Clave Ajena (salto modal)

Tal y como comentamos en la introducción de este punto, este tipo de salto responde a la necesidad de operar con una tabla asociada (típicamente a través de una clave ajena) en el transcurso de una operación de inserción/edición. gvHIDRA contempla este tipo de saltos desde su versión 4.0.0.

El ejemplo más claro es el de factura/proveedor. Imaginemos que un usuario que ha iniciado la introducción de una factura y, a mitad de introducción, cuando tiene que indicar el CIF del proveedor detecta que no está dado de alta en la BD. Esto, sin salto, supondría salir de la ventana, perder todos los datos insertar el proveedor y volver a empezar. Con el siguiente ejemplo, podremos insertar el proveedor sin necesidad de perder la información.

Hay varias consideraciones técnicas que no podemos obviar en este punto:

- Ventana modal: al no poder perder la edición del usuario, necesitamos abrir el salto en una ventana emergente. Esta debe ser modal para que el flujo de trabajo quede delimitado a lo que deseamos.
- Retorno con una acción de interfaz: al regresar a la ventana, querremos recoger el paso de mensajes y operar con ellos pero sin recargar la ventana. Para ello, necesitamos que el regreso del salto se realice a través de una acción de interfaz.
- Reutilización del mantenimiento: como hemos dicho, queremos poder operar con el mantenimiento de la clase correspondiente a la clave ajena, pero nos interesa poder aprovechar el mantenimiento ya existente.

Ficheros implicados:

Tabla 4.2. Ficheros implicados en implementación

clase manejadora	view	plantilla tpl	descripción
Facturas.php	p_facturas.php	p_facturas.tpl	Mantenimiento de facturas
Proveedores.php	p_proveedores.php	p_proveedores.tpl	Mantenimiento de proveedores.

Dando por hecho la creación de los dos mantenimientos vamos a ir, paso a paso, indicando como implementar el salto. En primer lugar, tenemos que ubicar un punto de disparo de la acción de salto en el origen del salto (p_facturas.tpl). A diferencia del ejemplo anterior, este salto sólo tiene sentido ser lanzado a través de un Botontooltip ubicado en el CWFicha o CWFila de la plantilla origen (p_facturas.tpl), por lo tanto, como se ha indicado arriba, el parámetro actuaSobre tendrá el valor del campo afectado:

```
{CWFicha}
  ...
  {CWCampoTexto nombre="ediCif" size="9" editable="true" textoAsociado="CIF"
 dataType=$smty_dataType_Facturas.ediCif}
  {CWBotonTooltip id="addProveedor" imagen="39" titulo="+"
 funcion="saltar" actuaSobre="ediCif"}
  ...
{/CWFicha}
```

Ahora, tenemos que crear en la clase manejadora origen del salto (Facturas), un método que gestione el salto (si se debe saltar o no; si pasa parámetros). En este caso el framework proporciona un método para tal uso **saltoDeVentana**:

```
public function saltoDeVentana($objDatos, $objSalto) {

    if($objSalto->getId()=='addProveedor') {
        $objSalto->setNameTargetClass('Proveedores');
        //Entramos en modo insercion
        $objSalto->setTargetAction('nuevo');
        //Indicamos que abra en modo modal estricto
        $objSalto->setModal(true,true);
        $objSalto->setSizeModal(1000,500);
        //Indicamos que debe volver lanzando la acción de interfaz.
        $objSalto->setReturnAsTriggerEvent(true);
```

```
        return 0;
    }
    return -1;
}
```

Ahora , configuraremos el objeto salto: para indicar la clase destino del salto, **setNameTargetClass()**. Con el método **setTargetAction()** fijamos la acción a ejecutar cuando realicemos el salto. Indicamos que se trata de un salto Modal (lo que hará que el navegador abra una ventana emergente) con el método **setModal(boolean,boolean), esta ventana modal la podemos redimensionar a nuestro gusto con el método setSizeModal()**. Y, finalmente, indicamos que queremos que al regresar ejecute una acción de interfaz con el método **setReturnAsTriggerEvent** (la acción a ejecutar corresponderá al id del salto: addProveedor).

setModal(boolean,boolean) El primer parámetro indicará si queremos que la ventana sea modal o se abra en la misma página del navegador, y el segundo, indicará si se trata de una ventana emergente modal estricta (true) o en cambio será una ventana emergente no estricta (false)

Nota: al realizar los saltos modales podemos encontrar problemas con algunos navegadores y el bloqueo de ventanas emergentes. Concretamente en Mozilla Firefox debemos comprobar que está permitido el lanzamiento de este tipo de ventanas para el servidor donde se ubique la aplicación.

Con esto, al pulsar el botón de salto addProveedor, el sistema realizará el salto a la ventana de mantenimiento de Proveedores entrando al modo de inserción directamente (estado nuevo). En este caso, nos interesa que el usuario inserte un nuevo proveedor y vuelva; es decir, que no pueda realizar ninguna operación más. Para controlar esto, utilizaremos la variable de presentación smty_modal en la tpl. Por ellos en la plantilla realizaremos estos cambios:

```
<!--*********** PANEL fil ******************-->
{*Mostramos el filtro y el tabular si hemos entrado a la ventana de forma
 normal*}
{*Cuando es modal, solo mostramos el modo edi que es donde realizará la
 inserción*}

{if $smty_modal neq 1}
    {CWPanel id="fil" action="buscar" method="post" estado="$estado_fil"
 claseManejadora="Proveedores"}
        {CWBarraSupPanel titulo="Mantenimiento de proveedores"}
    ...
{/if}
<!-- ***************** PANEL edi **********************-->
    {CWPanel id="edi" tipoComprobacion="envio"
 action="$smty_operacionFichaProveedores"
    method="post" estado="$estado_edi" claseManejadora="Proveedores"  accion=
$smty_operacionFichaProveedores}
    ...
        {CWBarraInfPanel}
            {CWBoton imagen="41" texto="Guardar" class="button"
 accion="guardar"}
            {*Si la abrimos como modal que al cancelar cierre la ventana. Sino
 que se comporte normal*}
            {if $smty_modal eq 1}
                {CWBoton imagen="42" texto="Cancelar" class="button"
 accion="cancelar" action="cancelarModal"}
            {else}
```

```
                {CWBoton imagen="42" texto="Cancelar" class="button"
 accion="cancelar" action="cancelarEdicion"}
            {/if}
        {/CWBarraInfPanel}
        ...
<!-- ****************** PESTAÑAS ************************-->
    {CWContenedorPestanyas}
        {*Si la abrimos como modal no nos interesa que se vea el filtro o el
 tabular*}
        {if $smty_modal neq 1}
            {CWPestanya tipo="fil" estado=$estado_fil}
            {CWPestanya tipo="lis" estado=$estado_lis}
        {/if}
        {CWPestanya tipo="edi" estado=$estado_edi}
    {/CWContenedorPestanyas}
```

Nota: Este cambio es opcional, es para ajustarse a las necesidades del ejemplo.

Arriba vemos que se definen dos botones (WCWBoton), uno para cuando la ventana es modal y otro para cuando no lo es. En el caso de encontrarnos que es una ventana modal hay que definir en el mappings la entrada correspondiente:

```
$this->_AddMapping('Proveedores__cancelarModal', 'Proveedores');
$this->_AddForward('Proveedores__cancelarModal', 'correcto', 'index.php?
view=views/p_proveedores.php&panel=listar');
```

Ahora tenemos que configurar la clase manejadora destino del salto (Proveedores). Básicamente tenemos que implementar el control del flujo cuando estamos en modo modal. En estos casos, tras realizar la operación, queremos que se cierre la ventana. Para ello tenemos que sobrecargar el método postInsertar (después de insertar, cerraremos la ventana) y la acción particular cancelarModal que hemos creado en la plantilla para cancelar el salto.

En el siguiente extracto tenemos el codigo correspondiente. Destacamos:

- **isModal**: método que indica si la ventana está en modo modal o no.
- **closeModalWindow**: método que lanza las instrucciones para cerrar la ventana modal.
- **setParam**: método del objeto salto para el paso de mensajes entre ventanas.

```
public function postInsertar($objDatos) {

    //Si se estamos en una ventana modal cerramos la ventana al salir
    if($this->isModal()) {
        $m_datosInsertados = $objDatos->getAllTuplas();
        $objSalto = IgepSession::dameSalto();
        $objSalto->setParam('nuevo',$m_datosInsertados[0]);
        $this->closeModalWindow();
    }
    return 0;
}

//Accion particular que cierra la ventana modal
public function accionesParticulares($str_accion, $objDatos) {
```

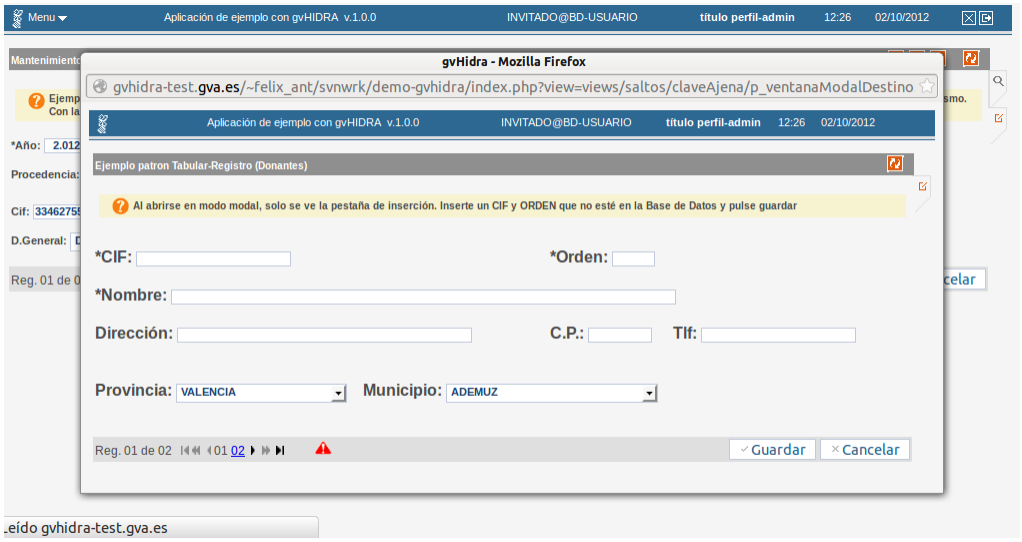
```
    if($str_accion=='cancelarModal') {

        $this->closeModalWindow();
        $fw = $objDatos->getForward('correcto');
        return $fw;
    }
}

/*************** REGRESO A VENTANA MODAL *********************/

public function regresoAVentana($objDatos, $objSalto) {

    $m_datosVisibles = $objDatos->getAllTuplas('visibles');
    $objSalto->setParam('nuevo',$m_datosVisibles[0]);
    return 0;
}
```

Finalmente, vemos que aparece el método **regresoAVentana**. Este método es el que se ejecutará para lanzar el regreso a la ventana. En él, vemos como se prepara un paso de parámetros con el método **setParam** (la identificiación del nuevo proveedor insertado).

Con esta implementación, al pulsar Guardar o Cancelar en la ventana modal, cerraremos dicha ventana y regresaremos a la ventana origen en el punto en el que la habíamos dejado ejecutando la acción de interfaz que asociemos al salto (addProveedor). Para lanzar esta acción de interfaz incorporamos en el constructor el comando y el método que resulve su ejeucción.

```
public function __construct() {
    ...
    $this->addTriggerEvent('addProveedor','cargarProveedor');
    ...
}
public function cargarProveedor($objDatos) {

    $salto = IgepSession::dameSalto();
    $proveedor = $salto->getParam('nuevo');
    if(!empty($proveedor['ediCif']) and !empty($proveedor['ediOrden'])) {
        $this->showMessage('APL-32',array($proveedor['ediCif'],
$proveedor['ediOrden']));
        $objDatos->setValue('ediCif',$proveedor['ediCif']);
        $objDatos->setValue('ediOrden',$proveedor['ediOrden']);
        $res = $this->consultar("select nombre as \"nombre\" from
 tinv_donantes ".
            "where cif = '".$proveedor['ediCif']."' and orden = ".
$proveedor['ediOrden']);
        if($res!=-1)
            $objDatos->setValue('ediNombre',$res[0]['nombre']);
    }
    IgepSession::borraSalto();
    return 0;
} //Fin de cargarProveedor
```

El resultado final es una ventana como la siguiente:



4.5. Carga dinámica de clases

4.5.1. Introducción

Hasta ahora todos los ficheros que usamos en una aplicación se cargan siempre independientemente de que se usen o no en el hilo actual de ejecución. Puesto que normalmente no se usa más de una clase de actions simultáneamente, esta situación se puede mejorar si cargamos las clases dinámicamente, y puede notarse especialmente en las aplicaciones grandes.

Desde la versión 5 de PHP, existe una funcionalidad nueva que permite, cuando no se encuentra una clase, llamar a una función de usuario donde se incluya el fichero correspondiente. Aprovechando esta funcionalidad nueva, se ha creado una clase en el framework que permite:

- **registerClass**: indicar cada clase en que fichero se ubica
- **registerFolder**: permite almacenar carpetas en las que buscar las clases (siempre que su nombre coincida con un fichero con extensión php)

Combinando los dos métodos podemos configurar la carga de clases de la manera más cómoda y eficiente teniendo en cuenta que:

- por defecto se registra la carpeta '*actions*'
- si una clase no se llama igual que el fichero que la contiene es necesario usar el método '*registerClass*'.

En cualquier caso, si no se desea utilizar esta funcionalidad basta con no invocar ninguno de los métodos anteriores. En el fichero include.php de la aplicación podemos seguir haciendo los includes de la manera tradicional.

4.5.2. Ejemplos de utilización

```
// obtenemos referencia al objeto
$al = GVHAutoLoad::singleton();

// registramos una carpeta donde tenemos clases
```

```
$al->registerFolder('actions/listados');

// si tengo pocas clases en una carpeta puedo optar por registrarlas
 individualmente
$al->registerClass('cabFactura', 'actions/factura/cabFactura.php');
$al->registerClass('linFactura', 'actions/factura/linFactura.php');

// si las clases no se llaman igual que el fichero donde se encuentran,
// no hay mas remedio que registrarlar individualmente
$al->registerClass('TinvTipos2',    'actions/TinvTipos.php');
$al->registerClass('TinvSubtipos2', 'actions/TinvSubtipos.php');

...
```

Parte III. Complementos al desarrollo

Tabla de contenidos

5. Fuentes de datos 148
5.1. Conexiones BBDD 148
5.1.1. Bases de Datos accesibles por gvHidra 148
5.1.2. Acceso y conexión con la Base de Datos 148
5.1.3. Transacciones 153
5.1.4. Procedimientos almacenados 155
5.1.5. Recomendaciones en el uso de SQL 156
5.2. Web Services 157
5.2.1. Web Services en PHP 157
5.2.2. Generación del WSDL 159
5.2.3. Web Services en gvHIDRA 160
5.3. ORM 162
5.3.1. Introducción 163
5.3.2. Ejemplo: Propel ORM 163
6. Seguridad 180
6.1. Autenticación de usuarios 180
6.1.1. Introducción 180
6.1.2. Elección del método de Autenticación 181
6.1.3. Crear un nuevo método de Autenticación 181
6.2. Modulos y Roles 181
6.2.1. Introducción 181
6.2.2. Uso en el framework 184
6.3. Permisos 185

Capítulo 5. Fuentes de datos

5.1. Conexiones BBDD

5.1.1. Bases de Datos accesibles por gvHidra

Una de las características importantes que tenemos en PHP es la gran cantidad de distintos tipos de bases de datos soportados. Cada una de ellas requiere de una extensión que define su propia API para realizar la misma tarea, desde establecer una conexión hasta usar sentencias preparadas o control de errores. Cambiar las llamadas a la API cuando cambiamos de BD en una aplicación es costoso pero se puede hacer, aunque en el caso de un framework no es factible porque tiene que funcionar el mismo código con varias BD. Lo que se suele hacer es añadir una capa de abstracción que nos permita operar de la misma manera con todos los tipos de BD (al menos para las operaciones más habituales o las más estandarizadas). En el caso de gvHIDRA se utiliza la librería MDB2 de PEAR (http://pear.php.net/package/MDB2/), una de las más usadas para este fin gracias a su amplia funcionalidad y a la actividad del proyecto. De esta forma podemos usar una BD sin preocuparnos por el tipo, ya que tenemos una API única.

Sin embargo, a pesar de usar la capa de abstracción, sigue habiendo aspectos que dependen del tipo de BD: uso de características especiales (secuencias, limit, bloqueos, ...), manejo de transacciones, representación de fechas y números, ... Para resolver estos problemas, definimos en el framework otra capa de abstracción (que se implementa en unas clases que se ven en el apartado siguiente) y las funciones más importantes que realizan son:

- Definición del formato de fechas y números que se va a usar para comunicarse con el gestor de BD. Este formato es transparente para el programador, ya que es el framework el que hace las conversiones necesarias cuando enviamos/recogemos información a/de la BD. Normalmente nos referiremos a éste como **formato de BD**.
- Inicialización de la conexión: son operaciones que se hacen cada vez que se hace una nueva conexión. Por ejemplo se fija la codificación que se va a usar en la comunicación (actualmente LATIN1 o ISO-8859-1, que es la usada en el framework), o se modifican parámetros de la conexión para establecer el formato de BD.
- Ajustes por sintaxis: aquí se incluyen una serie de métodos para obtener la sintaxis de ciertas operaciones que se pueden comportar de forma distinta en cada BD. Por ejemplo la clausula limit (para paginar resultados de consultas), función para concatenar cadenas, uso de secuencias, ...
- Unificar tratamiento de errores: por ejemplo, cuando hay un registro bloqueado, poderlo detectar independientemente de la BD.

Actualmente las BD soportadas por gvHIDRA son postgresql, mysql y oracle. Conforme vayamos soportando nuevos tipos (entre los soportados por MDB2), la funcionalidad de esta capa se podrá ir aumentando.

5.1.2. Acceso y conexión con la Base de Datos

Cómo crear conexiones a distintas fuentes de datos.

5.1.2.1. Introducción

El framework ofrece una serie de facilidades al programador que pueden llegar a ocultar las posibilidades que tiene de conexión a BBDD. Concretamente, tanto el pattern gvHidraForm_DB o el debug (IgepDebug), son ejemplos de uso de conexiones de forma transparente para el programador. En ambos casos, gvHidra

se encarga de conectar con el SGBD correspondiente y operar con él, dejando una serie de métodos que para consultar/operar.

Evidentemente, puede que nos encontremos con circunstancias que nos obliguen a utilizar algo más de lo que hemos expuesto. Para cubrir todo ello, el framework ofrece la clase IgepConexion que, a través de la librería PEAR::MDB2 conecta con diferentes SGBD. Esto permite crear conexiones a diferentes fuentes de forma homogenea e independizárnos de los drivers nativos que trabajan por debajo.

A esta capa de abstracción, el framework aporta una serie de perfiles de SGBD (IgepDBMS) que nos permiten configurar dichas conexiones para, por ejemplo, independizar el formato de la configuración del SGBD. Actualmente se han definido los siguientes perfiles:

Tabla 5.1. Perfiles SGBD

SGBD	PERFIL (CLASE)
Oracle	IgepDBMS_oci8.php
PostgreSql	IgepDBMS_pgsql.php
MySQL	IgepDBMS_mysql.php

Nota

se pueden crear perfiles para otros SGBD de forma sencilla.

5.1.2.2. Creación de una conexión

Para crear una conexión a una base de datos simplemente tenemos que crear una instancia de la clase IgepConexion pasándole el dsn de conexión a dicha BBDD. Dicho dsn tiene que tener la siguiente estructura (en PHP):

```
//Ejemplo postgresql
$dsnPos = array(
    'phptype'  => 'pgsql',
    'username' => 'xxxx',
    'password' => 'xxxx',
    'hostspec' => 'xxxx',
    'database' => 'xxxx',
);

//Ejemplo oracle
$dsnOra = array(
    'phptype'  => 'oci8',
    'username' => 'xxxx',
    'password' => 'xxxx',
    'hostspec' => 'xxxx',
);
```

Estos dsn se tienen que definir en el fichero de configuración de la aplicación gvHidraConfig.inc.xml siguiendo la sintaxis que especifica su DTD. También se pueden definir en la carga dinámica, aunque es poco recomendable.

La construcción de la conexión sería tan simple como:

```
//Nueva conexion a postgresql
$conPos = new IgepConexion($dsnPos);
```

```
//Nueva conexion a oracle
$conOra = new IgepConexion($dsnOra);
```

Después de crear la instancia, hay que comprobar si ha habido problemas de conexión:

```
if (PEAR::isError($conPos->getPEARConnection()))
     return;
```

¿Conexiones persistentes?

Si, conexiones persistentes aunque con matizaciones. El driver nativo de PHP para PostgreSQL ofrece la posibilidad de, con el objeto de optimizar, reutilizar las conexiones. De modo que, si creamos 10 conexiones, realmente tenemos 10 referencias a la misma conexión. Esto aumenta el rendimiento ya que sólo tenemos un "coste" de conexión aunque, evidentemente es peligroso. Por ello el framework, por defecto, crea conexiones independientes que mueren cuando el programador destruye el objeto (o cuando acaba el hilo de ejecución).

Puede que, en algunos casos puntuales, se quiera utilizar este tipo de conexiones (sólo funcionarían en PostgreSQL), y para esos casos se ha creado un parámetro en el constructor de la conexion (bool). Este parámetro, persistente, indica si el programador quiere solicitar una conexion persistente, siendo su valor por defecto false.

```
//Conexion permanente (reusable/reusada)
$pcon = new IgepConexion($dsn_log,true);
```

5.1.2.3. Utilizar conexión de la clase manejadora (gvHidraForm_DB)

Puede que en ciertas circunstancias nosotros queramos recuperar/utilizar la conexión que mantiene automáticamente la clase manejadora (las que extienden de gvHidraForm_DB). Para ello, se facilita el método getConnection que devuelve la instancia de la conexión en uso.

```
//En la clase manejadora
$con = $this->getConnection();
```

Nota

la clase manejadora ofrece también los métodos consultar y operar sobre esta conexión para poder lanzarlos sin recuperar la instancia de la conexión.

5.1.2.4. Métodos de conexión

En este apartado vamos a ver algunos de los métodos interesantes que ofrece la clase de conexiones de gvHIDRA. No los vermos todos (para ello consultar PHPDoc o doxy), pero si los más útiles.

5.1.2.4.1. consultar

Este método sirve para realizar una query con la conexión activa. El primer parámetro corresponde a la SELECT que queremos ejecutar. Opcionalmente podemos pasarle un segundo parámetro indicando si queremos que transforme los datos (especialmente fechas y números) al formato interno del framework. Sino se especifica este segundo parámetro, los datos vendrán en el formato configurado para cada SGBD, por lo que puede que no sean adecuados para operar con ellos.

```
// obtenemos datos para operar en capa de negocio (FW)
$res=$con->consultar(" SELECT fecha_ini,fecha_fin
            FROM tper_permisos
```

```
            WHERE nregpgv = '44908412R'",

 array( 'DATATYPES'=>array('fecha_ini'=>TIPO_FECHA,'fecha_fin'=>TIPO_FECHA,)));
```

5.1.2.4.2. operar

Este método sirve para lanzar operaciones DML más allá del comando SELECT; es decir UPDATE,INSERT o DELETE. En ocasiones necesita que se ejecute sobre los datos el método prepararOperacion.

5.1.2.4.3. prepararOperacion

Convierte los campos al formato que acepta la base de datos (escapar los carácteres especiales en textos, ajustar los separadores decimales y de grupos en números y formatos en fechas). Hay que llamarla antes del método consultar, operar o preparedQuery. Los parámetros que acepta son:

1. valor: puede ser un valor simple o un array de registros, donde cada registro es un array con estructura 'campo'=>valor. Es un parámetro por referencia, por lo que devuelve el resultado en esta variable.

2. tipo: si el valor es simple indicaremos directamente el tipo que puede ser las constantes: TIPO_CARACTER, TIPO_ENTERO, TIPO_DECIMAL, TIPO_FECHA y TIPO_FECHAHORA. Si el valor es un array, el tipo contiene un array asociativo con los tipos de cada columna. Más adelante se puede ver un ejemplo con valor simple, y éste seria un ejemplo con array:

```
$datos = array( array('nombre'=>"L'Eliana", 'superficie'=>'45.6') );
$conexion->prepararOperacion($datos,
 array('nombre'=>array('tipo'=>TIPO_CARACTER,),
 'superficie'=>array('tipo'=>TIPO_DECIMAL,),));
// en $datos ya tenemos los datos transformados
```

5.1.2.4.4. CalcularSecuenciaBD

Calcula el valor de una secuencia de BD. Consulte en el manual de su SGBD las opciones que ofrece al respecto.

5.1.2.4.5. CalcularSecuencia

Calcula el máximo de una tabla siguiendo varios criterios (según los parámetros).

1. tabla: Nombre de la tabla de la BD.

2. campoSecuencia: campo del que se quiere obtener la secuencia

3. camposDependientes: array que contiene el nombre de los campos de los cuales va a depender la secuencia y sus valores. Estructura [nombreBD] = valor

4. valorInicial: fija el valor inicial que devuelve calcularSecuencia en el caso de que no existan tuplas en la tabla. Valor por defecto 1.

```
public function calcularSecuencia($tabla, $campoSecuencia,
 $camposDependientes, $valorInicial=1)
```

5.1.2.4.6. preparedQuery

Es similar a consultar y operar, pero usando sentencias preparadas. Es más eficiente que estar formando cada vez la sentencia SQL ya que el SGBD solo tiene que analizar la sentencia la primera vez. No requiere escapar caracteres especiales. Es recomendable especialmente cuando usamos la sentencia dentro de

bucles. Al igual que consultar, también tiene un parámetro opcional, que permite pasarle los tipos de los campos. El último parámetro también es opcional y sirve para conservar la sentencia preparada cuando ejecutamos la misma sentencia varias veces con distintos parámetros.

A continuación ponemos un ejemplo de utilización. Comprobamos que, dadas las facturas del año 2006, todas sus líneas esten en estado 'REVISADA', y si no es así actualizamos la factura en concreto pasándola a estado 'SINREVISION' y le añadiremos el campo observaciones que ha introducido el usuario.

```
$conexion = new IgepConexion($dsn);
$sel = "SELECT nfactura as \"nfactura\" FROM lineas WHERE anyo='".2005."' AND
 estado<>'REVISADA' group by nfactura";
$res = $conexion->consultar($sel);
if ($res!=-1 and count($res[0])>0) {
   //Como no sabemos el contenido de observaciones escapamos por si hay
 cualquier caracter problematico
   $conexion->prepararOperacion($observaciones, TIPO_CARACTER);
   foreach ($res as $facturaSinRevision){
      $res2 = $conexion->operar("UPDATE factura set
 observaciones='$observaciones', estado='SINREVISION'
                                WHERE anyo='2005' AND factura='".
$facturaSinRevision['nfactura']."'");
      if ($res2==-1) {
         // Mensaje de Error
         ....
      }
}
```

El mismo ejemplo usando sentencias preparadas seria:

```
$conexion = new IgepConexion($dsn);
try {
   $sel = "SELECT nfactura as \"nfactura\" FROM lineas WHERE anyo=? AND
 estado<>'REVISADA' group by nfactura";
   $res = $conexion->preparedQuery($sel, false, array(2005));
   if (count($res[0])>0) {
      $upd = null;
      $sql = "UPDATE factura set observaciones=?, estado='SINREVISION' WHERE
 anyo=? AND factura=?";
      //con sentencias preparadas no es necesario escapar caracteres
 especiales
      foreach ($res as $facturaSinRevision)
         $res2 = $conexion->preparedQuery($sql, true,
 array($observaciones,'2005',$facturaSinRevision['nfactura']), null, $upd);
   }
} catch (Exception $e) {
   // Mensaje de Error
   ...
}
```

El control de errores con sentencias preparadas siempre es a través de excepciones [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/excepciones.html].

También están disponibles los métodos consultarForUpdate y preparedQueryForUpdate que permiten bloquear los registros que recuperan. Más información aquí [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/trans.html].

5.1.2.5. Utilizar el driver MDB2 o nativo

En algunos casos, puede ser interesante trabajar directamente con el driver PEAR::MDB2 o el driver nativo de PHP. Un ejemplo puede ser la utilización de procedimientos almacenados. Para estos casos el FW ofrece el método getPEARConnection que devuelve el objeto PEAR::MDB2 conectado

```
/*CONEXION ACTIVA DE LA CLASE MANEJADORA*/
//Recogemos el objeto IgepConexion de la conexion activa.
$conPEARMDB2 = $this->getConnection();
//Recogemos el objeto PEAR::MDB2 de la conexion activa.
$conPEARMDB2 = $this->getConnection()->getPEARConnection();
//Recogemos el driver nativo de PHP de la conexion activa.
$conPHP = $this->getConnection()->getPEARConnection()->getConnection();

/*NUEVA CONEXION*/
//Recogemos el objeto PEAR::MDB2
$this->_conOracle = new IgepConexion($g_dns_ora);
$conPEARMDB2 = $this->_conOracle->getPEARConnection();
//Recogemos el driver nativo de PHP
$conPHP = $this->_conOracle->getPEARConnection()->getConnection();
```

5.1.3. Transacciones

5.1.3.1. Introducción

Actualmente el framework no fija el parámetro autocommit, por lo que estamos usando el valor fijado en los servidores: activado para postgres y mysql, y desactivado para oracle (de hecho, no se puede activar). En próximas versiones se normalizará esta situación. Cuando queramos englobar varias operaciones en una transacción, tendremos que iniciar una transacción o asegurarnos que ya estamos en una. De momento no hay soporte para transacciones anidadas.

Nota

En mysql, para tener soporte transaccional hay que definir las tablas con storages que lo soporten como InnoDB.

El framework inicia una transaccion automaticamente para hacer las operaciones del CRUD. Para el resto de casos, deberemos iniciar ésta de modo explícito. Se puede hacer con el método **empezarTransaccion** de IgepConexion. En las clases de negocio no hace falta hacerlo (ya que el framework inicia la transacción), a menos que queramos que la transacción empiece en preInsertar, preModificar o preBorrar.

Ejemplo:

```
$this->getConnection()->empezarTransaccion();
```

También es habitual empezar una transacción cuando tenemos una conexión sobre un DSN distinto al empleado en el formulario, sobre el que queremos hacer operaciones de actualización.

La transacción acaba con el método **acabarTransaccion**, y recibe un parámetro booleano indicando si ha habido error, y en función de éste se hace un commit o rollback. No hay que llamarlo cuando se trata de la transacción del framework.

5.1.3.2. Control de Concurrencia

Para las operaciones del CRUD, el framework aplica un método simple para evitar que dos usuarios modifiquen simultáneamente un registro que previamente han leído. Consiste en añadir a la condición del where, en los update y delete, todos los campos modificables de la tabla con los valores leídos previamente. Si el registro no se encuentra significa que otro usuario lo ha modificado, y entonces se obliga al usuario a repetir la operación.

Esta aproximación también puede usarse en otros casos, aunque cuando la transacción incluye muchas operaciones, puede ser muy complejo. Para estas situaciones, también se puede usar la opción de bloquear los registros, como se explica a continuación.

5.1.3.2.1. Bloqueos

Las operaciones DML update y delete bloquean implícitamente los registros afectados. Sin embargo hay ocasiones en las que primero leemos información de la BD, y en función de ella actualizamos otra información. Ejemplos de estos casos son al incrementar un saldo, obtener un numero secuencial, ... Para estas situaciones nos interesa bloquear los registros que leemos, y así nos aseguramos que no van a cambiar mientras hacemos la actualización.

En las conexiones con autocommit, hay que tener la precaución de empezar una transacción antes de hacer los bloqueos (o asegurarnos que estamos en una transacción ya empezada). Los bloqueos se liberan cuando la transacción finaliza. También hay que recordar que al finalizar cada petición al servidor, se cierran las conexiones con las bases de datos, y por tanto también se liberan todos los bloqueos. Eso significa que por ejemplo, no podemos mantener bloqueado un registro mientras el usuario lo edita.

Los bloqueos se hacen sin espera (si el registro está bloqueado por otro usuario se produce un error), con el fin de no colapsar el servidor web. En el caso de mysql no se dispone de la opción de no esperar, aunque la espera se corta tras un timeout. Este timeout por defecto es de 50 segundos, por lo que conviene cambiar el parámetro innodb_lock_wait_timeout a un valor de menos de 5 segundos.

Para hacer estos bloqueos, tenemos en IgepConexion el método **consultarForUpdate** que funciona exactamente como consultar pero bloqueando los registros afectados. Si algún registro ya esta bloqueado por otro usuario, se produce una excepción que habrá que capturar. A continuación tenemos un ejemplo de su utilización:

```
$con->empezarTransaccion();
try {
   $res = $con->consultarForUpdate('select * from tinv_donantes where
 orden=5');
   if ($res == -1) {
      $this->showMessage('APL-x'); // algun problema con la consulta
      $con->acabarTransaccion(1);
      return -1;
   }
} catch (gvHidraLockException $e) {
   $this->showMessage('APL-y'); // algun registro bloqueado
   $con->acabarTransaccion(1);
   return -1;
}
// actualizaciones varias relacionadas con el registro que tenemos bloqueado
//...
$con->acabarTransaccion(0);
```

También podemos hacer lo mismo usando el método **preparedQueryForUpdate**, que funciona similar pero usando sentencias preparadas:

```
$con->empezarTransaccion();
try {
   $res = $con->preparedQueryForUpdate('select * from tinv_donantes where
 orden=?', array(5,));
} catch (gvHidraLockException $e) {
   $this->showMessage('APL-x'); // algun registro bloqueado
   $con->acabarTransaccion(1);
   return -1;
} catch (gvHidraSQLException $e) {
   $this->showMessage('APL-y'); // algun problema con la consulta; con $e-
>getSqlerror() obtenemos error PEAR
   $con->acabarTransaccion(1);
   return -1;
}
// actualizaciones varias relacionadas con el registro que tenemos bloqueado
//...
$con->acabarTransaccion(0);
```

En el capítulo 7 [188] se explican con más detalle las excepciones que se pueden utilizar.

En la consulta usada para bloquear conviene limitar al máximo el numero de registros obtenidos a los estrictamente necesarios, y no emplear joins ya que se bloquearían los registros afectados de todas las tablas. De esta manera aumentamos el nivel de concurrencia y reducimos la posibilidad de error por registros bloqueados.

5.1.4. Procedimientos almacenados

En este apartado intentaremos explicar como utilizar procedimientos almacenados en gvHIDRA

5.1.4.1. Introducción

Puede que nuestro sistema de información se haya diseñado depositando toda la lógica de la aplicación en procedimientos almacenados o que, puntualmente, tengamos la necesidad de acceder a uno de esos recursos del SGBD. Para ellos, en este apartado daremos algunos ejemplos de como hacerlo a través de las conexiones del FW.

5.1.4.2. Creación de una conexión

Las conexiones al framework están organizadas en tres capas (capa DBMS-gvHIDRA, capa MDB2 y driver nativo PHP). Las llamadas a procedimientos almacenados se pueden realizar desde cualquiera de las capas, pero ganamos en versatilidad a medida que nos acercamos al driver nativo.

La capa DBMS-gvHIDRA actualmente sólo te permite trabajar mediante el método operar, lo cual, excepto para casos muy puntuales, no parece cómodo. La capa MDB2 tiene métodos como el **executeStoredProc** especialmente para estos casos. De todos modos, nuestra recomendación es utilizar el driver nativo ya que parece que en los dos casos anteriores no se trabaja bien con valores entrada/salida.

```
$conexion = new IgepConexion($dsn);
$con = $conexion->getPEARConnection()->getConnection();
```

```
$query="begin proced('1234', :retorno, :msgproc); end;";
$id_result= @OCIParse( $con, $query);
if (!id_result or OCIError($id_result))
        return;
OCIBindByName ( $id_result, ":retorno", $ret, 10 );
OCIBindByName ( $id_result, ":msgproc", $msg, 200 );
@OCIExecute ( $id_result, OCI_DEFAULT );//ATENCION A partir de PHP 5.3.2
 cambiar OCI_DEFAULT por OCI_NO_AUTO_COMMIT
```

5.1.5. Recomendaciones en el uso de SQL

En este documento se dan algunas recomendaciones para que las sentencias SQL usadas en gvHidra sean lo más estándar posible, y minimizar el impacto de un posible cambio de gestor de base de datos (SGBD). Los SGBD considerados son PostgreSQL, Oracle y Mysql.

5.1.5.1. Alias en las columnas

Se recomienda poner siempre alias en los campos de una consulta.

Si no usamos los alias, en oracle los nombres de las columnas siempre saldrán en mayúsculas, con lo que no funcionará si cambiamos o otro SGBD. En PEAR::MDB2 hay una opción de compatibilidad que transforma todo a minúscula, pero como no permite tener identificadores mixtos (del tipo 'NombreProveedor'), se ha optado por no habilitarla. Para que el alias tenga efecto hay que ponerlo con comillas dobles:

```
select dpro as "nombre"
from provincias
```

5.1.5.2. Comprobación de nulos

No usar NVL ni DECODE en oracle ya que no existen en Postgresql ni en Mysql. Mejor usar el case que funciona en todos y es estándar SQL (el coalesce es más parecido al nvl y también es estándar, aunque no funciona con oracle 8).

```
SELECT CASE WHEN cpro is null THEN '-' ELSE cpro END
FROM tcom_usuarios
```

5.1.5.3. Concatenaciones

Las concatenaciones se hacen con la función 'concat(str1,str2)' (estándar, en Mysql y oracle ya existe, en postgresql se crea (con el script a continuación); el operador '||' funcionaba en oracle y postgres aunque no se recomienda).

```
CREATE OR REPLACE FUNCTION concat(text, text)
  RETURNS text AS
$BODY$
  BEGIN
    RETURN coalesce($1,'') || coalesce($2,'');
  END;
$BODY$
  LANGUAGE 'plpgsql' VOLATILE;

GRANT EXECUTE ON FUNCTION concat(text, text) TO public;
```

5.2. Web Services

- Web Services en PHP [157]
  - Cliente [157]
  - Servidor [157]
- Generación del WSDL [159]
- Web Services en gvHIDRA [160]
  - Cliente [160]
  - Servidor [161]

5.2.1. Web Services en PHP

5.2.1.1. Cliente

La creación de un cliente de un servicio web con PHP es relativamente sencilla haciendo uso de PHP-SOAP. Con la descripción del servicio al que queremos acceder (fichero wsdl), obtendremos acceso a todos los métodos que ofrece el servicio web. A continuación mostramos un ejemplo donde se verá más claramente lo expuesto. Concretamente en el ejemplo llamamos a un WS que, dada una cadena, devuelve la cadena al revés.

```
$objClienteWS = new SoapClient('Ws_Ejemplo.wsdl');
$resultado = $objClienteWS->ejemplo('Hola');
print_r($resultado);
```

De la ejecución de este cliente obtenemos el siguiente resultado:

```
  aloH
```

5.2.1.2. Servidor

La creación del servidor requiere, evidentemente, algo más de trabajo que la del cliente. En este punto haremos un pequeño resumen de los pasos a seguir. Primero tenemos que crear un fichero php (en nuestro ejemplo server.php) que contendrá las llamadas a las clases SOAP correspondientes al servidor. En este mismo fichero se puede incluir la definición de la clase que implementará todos los métodos exportados. Siguiendo con nuestro ejemplo, tenemos que tener un método que nos devuelva la inversa de una cadena. El contenido del fichero es:

```
require_once 'SOAP/Server.php';

 class Prueba_Server {
    function ejemplo($cadena){
      return strrev($cadena);
    }
 }

 $server = new SOAP_Server;
 $server->_auto_translation = true;
```

```
 $soapclass = new Prueba_Server();
 $server->addObjectMap($soapclass,'urn:Prueba_Server');
 $server->service($HTTP_RAW_POST_DATA);
```

Para que el cliente tenga acceso a la información que ofrece el WS, necesita de la definición de los métodos exportados por la clase. Esto se obtiene a partir del fichero WSDL. El fichero de nuestro ejemplo es el siguiente:

```
<?xml version="1.0"?>
<definitions name="ServerExample" targetNamespace="urn:ServerExample"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="urn:ServerExample"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

<types xmlns="http://schemas.xmlsoap.org/wsdl/"></types>
<message name="ejemploRequest">
  <part name="cadena" type="xsd:string" />
</message>
<message name="ejemploResponse">
  <part name="cadena" type="xsd:string" />
</message>
<portType name="ServerExamplePort">
  <operation name="ejemplo">
    <input message="tns:ejemploRequest" />
    <output message="tns:ejemploResponse" />
  </operation>
</portType>
<binding name="ServerExampleBinding" type="tns:ServerExamplePort">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="ejemplo">
    <soap:operation soapAction="urn:Prueba_Server#prueba_server#ejemplo" />
     <input>
       <soap:body use="encoded" namespace="urn:Prueba_Server"
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
     </input>
     <output>
       <soap:body use="encoded" namespace="urn:Prueba_Server"
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
     </output>
  </operation>
</binding>
<service name="ServerExampleService">
 <documentation />
  <port name="ServerExamplePort" binding="tns:ServerExampleBinding">
   <soap:address location="http://mi-servidor.es/cin/ws/server/server.php" />
  </port>
</service>
</definitions>
```

Con esto nuestro Web Service ya esta funcionando. Simplemente tenemos que llamarlo desde el cliente.

5.2.2. Generación del WSDL

Uno de los puntos más costosos de la creación de un WS en PHP es la generación del WSDL. Hay una serie de herramientas que te permiten su generación siempre y cuando rellenes ciertos parámetros. El propio PEAR::SOAP te facilita un proceso para hacerlo, y es el que vamos a intentar explicar a continuación.

La idea consiste en crear un constructor para nuestra clase y, en él, sobrescribir una serie de arrays donde aportaremos la información de los métodos que exporta una clase. Concretamente se trata de la estructura $this->__dispatch_map que tendrá una entrada por cada uno de los métodos soportados. Después, tras las llamadas que realizamos al SOAP_Server, tenemos que utilizar una clase de SOAP que nos generará el WSDL.

El código para nuestro ejemplo es el siguiente:

```
require_once 'SOAP/Server.php';

class Prueba_Server {
   function Prueba_Server(){
       $this->__dispatch_map['ejemplo'] =
           array(
               'in' => array('cadena' => 'string',
                       ),
               'out' => array('cadena' => 'string'),
           );
   }
   function ejemplo($cadena){
     return strrev($cadena);
   }
}


$server = new SOAP_Server;
$server->_auto_translation = true;
$soapclass = new Prueba_Server();
$server->addObjectMap($soapclass,'urn:Prueba_Server');

if(isset($_REQUEST['wsdl'])){
  require_once 'SOAP/Disco.php';
  $disco = new SOAP_DISCO_Server($server,'ServerExample');
  header("Content-type: text/xml");
  echo $disco->getWSDL();
  return;
}

$server->service($HTTP_RAW_POST_DATA);
```

Con este código, al interrogar al server.php directamente, obtendremos un xml que contiene la definición del WS. Este contenido se almacena en un fichero WSDL y, de ese modo, los clientes podrán acceder al servicio.

5.2.3. Web Services en gvHIDRA

Ya hemos visto como se implementan los WS en PHP. Para aplicaciones con gvHidra, se han implementado algunas clases que ayudan a la creación tanto del servidor como del cliente en una aplicación. De momento se utiliza un procedimiento muy básico para controlar la seguridad.

5.2.3.1. Cliente

La creación del cliente se realiza mediante la clase IgepWS_Client. A continuación mostramos el código:

```
$objCIN = IgepWS_Client::getClient('actions/ws/WSCIN.wsdl');
$credencial = IgepWS_Client::getCredential('WSCIN');
$objCIN->getImporte($credencial,....
```

El parámetro credencial contiene los parámetros de validación válidos para ese cliente WS. Este parámetro se obtiene consultando los DSNs de la aplicación. La estructura de la credencial seria la siguiente:

```
$credencial = array(
               'login'=>'XXXXX',
               'password'=>'XXXXX'
   );
```

Esta clase usa SoapClient, y si necesitamos pasarle otras opciones podemos hacerlo con el parámetro opcional. En el ejemplo siguiente se modifican las opciones para poder depurar la comunicación con el servidor. En la web de PHP [http://www.php.net/manual/en/soapclient.construct.php] se pueden ver otras opciones posibles.

```
$obj = IgepWS_Client::getClient('actions/ws/WSCIN.wsdl', array("exceptions" =>
 0, 'trace'=>1, ));
print "<pre>\n";
print "Request :       \n". htmlspecialchars($obj->__getLastRequest())  ."\n";
print "RequestHeaders :\n". $obj->__getLastRequestHeaders()             ."\n";
print "Response:       \n". htmlspecialchars($obj->__getLastResponse()) ."\n";
print "ResponseHeaders:\n". $obj->__getLastResponseHeaders()            ."\n";
print "Functions:      \n". var_export($obj->__getFunctions(),true)     ."\n";
print "Types:          \n". var_export($obj->__getTypes(),true)         ."\n";
print "</pre>";
```

Para depuración, también puede ser útil inhabilitar la cache del cliente, ya que así podemos ir modificando el wsdl:

```
IgepWS_Client::disableCache();
```

Si el cliente usa una codificación distinta a UTF-8 (como ocurre con gvHIDRA que usa latin1), conviene indicarlo en las opciones de conexión y así no hay que hacer conversiones explicitas, sino que PHP convierte los parámetros de entrada desde la codificación origen a UTF-8, y el resultado lo convierte de UTF-8 a la codificación indicada. Si no lo indicamos se fija a 'latin1'.

Ejemplo:

```
$obj = IgepWS_Client::getClient('actions/ws/WSCIN.wsdl', array("exceptions" =>
 0, 'trace'=>1, ));
...
print "Response:\n".htmlspecialchars(utf8_decode($obj-
>__getLastResponse())))."\n";
```

```
...
```

5.2.3.2. Servidor

En el caso del servidor, los beneficios son mayores. Tenemos dos clases, una clase estática que genera el código básico del Servidor y otra que proporciona un comportamiento al servidor propio de una aplicación gvHidra. La primera de ellas es la clase estática IgepWS_ServerBase. Esta clase simplifica la creación y el registro de un servidor SOAP de WS. Concretamente, en nuestro fichero de lanzamiento del servidor (típicamente server.php), que tiene que estar en el raiz de la aplicación, tendríamos el siguiente código:

```
 require_once "igep/include/igep_ws/IgepWS_ServerBase.php";
 require_once 'ws/server/WSCIN.php';

 IgepWS_ServerBase::registrar('WSCIN');
```

Por otro lado, tenemos que crear la clase que controla el Servidor (en nuestro ejemplo WSCIN). Esta clase, al heredar de IgepWS_Server tiene el mecanismo de validación ya implementado y el sistema de conexión propio de gvHidra. La única premisa que se exige es que, si se requiere validación, el método implementado por el programador debe incluir un parámetro $credencial que se pasará al método checkCredential para validar su contenido.

A continuación mostramos un ejemplo:

```
include_once "igep/include/igep_ws/IgepWS_Server.php";

class WSCIN extends IgepWS_Server
{
    function __construct()
    {
        $msgs = array(
                '1'=>'Error de conexión. Avise al Servicio de Informática',
                '2'=>'Error en operación. Avise al Servicio de Informática',
        );
        parent::__construct('WSCIN', $msgs);
        ...
    }

    function getImporte($credencial,$anyox, $dgralx, $numx, $tipo_expx,
 $numtipo_expx)
    {
      if (!$this->checkCredential($credencial))
        return $this->getAuthError();
      $dsn = ConfigFramework::getConfig()->getDSN('dsn_cin');
      $db = $this->conectar($dsn);
      if (!$db)
        return $this->gvhFault('1');
      ...
      return array('implic' => floatval($res[0]['implic']), 'impadj' =>
 floatval($res[0]['impadj']));
    }
    ...
}
```

Si nuestro servidor acepta varias credenciales, podemos pasarle un vector al constructor del padre:

```
parent::IgepWS_Server(array('WSCIN','WSMCMENOR',), $msgs);
```

En las credenciales de los servidores de web services, la contraseña hay que almacenarla con hash. Para ello usar el formulario en igep/include/igep_utils/protectdata.php para obtener los hash de las contraseñas, y guardar estas últimas en un lugar seguro, fuera de la aplicación.

El parámetro con los mensajes de error se utiliza si provocamos los errores (Soap_Fault) con el método gvhFault, que hace las conversiones necesarias en la codificación y tiene un parámetro opcional usado para que nos informe del error en el debug (tener en cuenta que los errores de conexión y los del método consultar ya se registran en el debug). Por ejemplo:

```
return $this->gvhFault('2','Error consultando tabla');
```

Si queremos restringir algún método a algunas credenciales, podemos hacerlo con método checkCredential pasándole la lista de credenciales:

```
if (!$this->checkCredential($credencial, array('WSMCMENOR',)))
        return $this->getAuthError();
```

También hay que tener precaución cuando hacemos consultas sobre base de datos, que si utilizamos campos calculados o agregados (count, min, ...), el resultado será de tipo string. Si queremos obtener otro tipo tendremos que modificarlo usando la función floatval para tipo float, intval para tipo int, ... En caso de problemas podemos forzar las conversiones con usando la clase SOAP_Value donde básicamente le indicamos el nombre del atributo, el tipo y el valor.

Ejemplo:

```
$modulos_usuario = new SOAP_Value('modulos', 'ModulosValidaStruct',
 $array_con_modulos);
```

También es importante tener en cuenta la codificación, sobretodo cuando nuestro WS recibe o devuelve campos string que puedan tener carácteres especiales.

Cuestiones:

- la codificación usada en los WS es UTF-8, luego habrá que hacer las transformaciones necesarias desde/hacia LATIN-1 (la codificación usada en gvHIDRA)
- si queremos retornar una cadena obtenida de la BD o de una constante en un fichero fuente de PHP, tenemos que transformarla a utf-8 con utf8_encode($cadena), o mejor usamos el método encode.
- si recibimos un parámetro texto vendrá en utf-8, luego también habrá que transformarlo (utf8_decode) a latin-1 para operar con él (concatenar con otras cadenas, almacenar en BD, ...). Lo podemos hacer con el método decode.
- en caso de problemas también podemos hacer las transformaciones con iconv [http://www.php.net/iconv].

En caso de problemas con algún tipo de datos conviene consultar la documentación en http://www.w3.org/TR/xmlschema-2.

5.3. ORM

En este apartado vamos a hablar de como integrar un ORM en el framework gvHIDRA. Actualmente, no se ha desarrollado una clase específica para uso de ORMs, pero es, como veremos en este apartado, sencilla su integración.

5.3.1. Introducción

Entendemos un ORM (mapeo objeto-relacional) como un sistema o técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Utilizar un ORM tiene una serie de ventajas que nos facilitan enormemente tareas comunes y de mantenimiento:

- **Reutilización**: La principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.
- **Encapsulación**: La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.
- **Portabilidad**: Utilizar una capa de abstracción nos permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no utilizamos una sintaxis MySQL, Oracle o SQLite para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.
- **Seguridad**: Los ORM suelen implementar mecanismos de seguridad que protegen nuestra aplicación de los ataques más comunes como SQL Injections.
- **Mantenimiento del código**: Gracias a la correcta ordenación de la capa de datos, modificar y mantener nuestro código es una tarea sencilla.

5.3.2. Ejemplo: Propel ORM

Actualmente existen muchas soluciones ORM con PHP en el mercado, tanto libres como propietario, aunque para este ejemplo hemos optado por un enfoque Open-Source con el objetivo de seguir con la misma dirección que gvHidra. Como se ha indicado anteriormente, cualquier otra solución sería válida.

Propel es un ORM basado en PHP. Mediante el mapeo de una BBDD en XML se generan un conjunto de clases para su manejo, simplificando de este modo el acceso a los datos, dando una solución CRUD.

5.3.2.1. Características

Según indica la propia documentación oficial de Propel, funciona con los siguientes SGBD's:

- PostgreSQL
- MySQL
- Oracle
- Sql Server
- Sqlite

El trabajo con las clases generadas es cómodo e intuitivo siempre que se tengan nociones de desarrollo con objetos. En un principio aunque la curva de aprendizaje puede parecer mas compleja de lo esperado, por las pruebas realizadas el resultado mediante este ORM es rápido y eficaz. Realmente no cuesta un

tiempo excesivo hacer una consulta o inserción en las tablas de la BBDD. Las opciones que nos brinda este ORM son muchas y variadas; van desde realizar consultas sin necesidad de utilizar SQL, hasta controlar la consulta que se va a realizar contra la BD introduciendo una sentencia SQL mas compleja.

El tratamiento de los datos devueltos en las consultas también es un punto importante, ya que se puede recuperar un solo elemento de una consulta, o en caso de necesitarlo, un array de objetos que contienen esa información. Eso significa que de una sola consulta, el ORM nos permite recuperar un conjunto finito de resultados, un conjunto finito de objetos de una clase.

Propel nos da una enorme variedad de opciones, éstas nos cubrirán en la mayoría de los casos aquellas funciones o necesidades que tengamos. Gracias a la herencia, si alguno de los casos no estuviera cubierto como debiera, entonces tenemos la opción de crearnos nuestra propia clase de gestión de los datos, con las mejoras que consideremos oportunas. Recordemos que disponemos de todas las ventajas de la programación orientada a objetos, como la sobrecarga o la herencia.

También nos da una gran variedad de operaciones, aparte de las básicas de CRUD. A continuación se detallan algunas de las más importantes:

1. *Inserción de filas relacionadas*
2. *Guardado en cascada*
3. *Uso de las relaciones en una consulta*
   - Búsqueda de registros relacionados con otro
   - Consultas embebidas
   - Relaciones uno a uno
   - Relaciones muchos a muchos
   - Reducir al mínimo las consultas
4. *Validaciones*
   - Igual que
   - No igual que
   - Tamaño máximo
   - Tamaño mínimo
   - Tipo
   - Valor válido
   - Personalizadas
5. *Transacciones*: Las transacciones de bases de datos son la clave para asegurar la integridad de los datos y el rendimiento de las consultas de base de datos.
6. *Herencia*
   - Igual que
   - No igual que

5.3.2.2. Configuración

La instalación se puede hacer mediante **PEAR**, es importante tener en cuenta que tendremos que haber instalado previamente también Phing, una vez hecho esto, se accede a los canales correspondientes y se instalan los paquetes de Propel.

La fase previa al uso de Propel en la cual se definen las estructuras de datos y las clases manejadoras, se puede abordar de dos modos:

5.3.2.2.1. Desde un XML

Se generan a través de funciones nativas de Propel y mas concretamente "*propel-gen*", una serie de clases manejadoras de la BBDD, tanto las clases base, como unas clases que heredan de estas primeras

Tendremos entonces que generar el SQL para poder crear las tablas en la BBDD, este proceso también es automatizado mediante la herramienta "propel-gen", que nos generará un fichero SQL conteniendo toda la estructura de la BBDD, este fichero lo usaremos para crear en la BBDD las tablas correspondientes.

5.3.2.2.2. Mediante Ingeniería InversaL

Obtendremos el XML con toda la estructura de las tablas implicadas.

Realizamos entonces los mismos pasos que en el primer caso, para poder generar la estructura de las clases manejadoras.

5.3.2.3. En marcha

La instalación se puede hacer mediante **PEAR**, es importante tener en cuenta que tendremos que haber instalado previamente también Phing, una vez hecho esto, se accede a los canales correspondientes y se instalan los paquetes de propel.

Pasos para realizar esta instalación:

1. Se debe agregar el canal pear.propelorm.org al entorno de PEAR.
2. Una vez que tenemos el canal, puede instalar el paquete del generador, o el paquete de tiempo de ejecución, o ambas cosas, si es una primera instalación, preferentemente instalar ambas cosas.
3. Es importante hacer uso de la opción '-a' para permitir que PEAR pueda descargar e instalar las dependencias.

```
pear channel-discover pear.propelorm.org
pear install -a propel/propel_generator
pear install -a propel/propel_runtime
```

El generador de Propel utiliza Phing 2.3.3, si no esta instalado, se debe instalar.

Pasos para realizar esta instalación:

1. Se debe agregar el canal *pear.phing.org* al entorno de PEAR.
2. Posteriormente instalamos el *Phing* y el *Log*.

```
pear channel-discover pear.phing.info
pear install phing/phing
pear install Log
```

Una vez realizadas las instalaciones, simplemente tendremos que hacer uso de “propel-gen” para comprobar que todo ha funcionado correctamente, aunque el proceso no realice ninguna función, ya que no habremos creado los ficheros necesarios para ello.

Podemos recuperar la información, los datos, de la base de datos y prepararla en un fichero XML, para ello Propel tiene herramientas que nos lo permitirán, generándonos un fichero XML que contiene todos estos datos.

Posteriormente se explicaran los pasos previos a seguir para hacer uso de Propel. Antes de poder comenzar a usar Propel, tendremos que preparar la capa de comunicación con nuestra BBDD para lo cual nos vamos a encontrar con dos situaciones:

CASO 1: Partimos de una BBDD en formato XML.

1. Se generan a través de funciones nativas de Propel y mas concretamente “propel-gen”, una serie de clases manejadoras de la BBDD, tanto las clases base, como unas clases que heredan de estas primeras.

2. Tendremos entonces que generar el SQL para poder crear las tablas en la BBDD, este proceso también es automatizado mediante la herramienta “propel-gen”, que nos generará un fichero SQL conteniendo toda la estructura de la BBDD, este fichero lo usaremos para crear en la BBDD las tablas correspondientes.

CASO 2: Partimos de una BBDD ya creada, usamos ingeniería inversa.

1. Obtendremos el XML con toda la estructura de las tablas implicadas.

2. Realizamos entonces los mismos pasos que en el primer caso, para poder generar la estructura de las clases manejadoras.

Se detalla a nivel técnico cada uno de los dos casos, con su comportamiento correspondiente.

CASO 1: BBDD en formato XML

1. Crearemos un nuevo fichero llamado *schema.xml* que contendrá la estructura de nuestra BBDD siguiendo el siguiente esqueleto:

```
<?xml version="1.0" encoding="UTF-8"?>
 <database name="" defaultIdMethod="native">
   <table name="" phpName="">
      <!-- column and foreign key definitions go here -->
      <column name="" type="integer" required="true" primaryKey="true"
 autoIncrement="true"/>
       ...
      <foreign-key foreignTable="" phpName="" refPhpName="">
        <reference local="" foreign=""/>
      </foreign-key>
     …
   </table>
 </database>
```

En este fichero XML vamos a incluir toda la estructura de la BBDD con la que queremos trabajar, las columnas y las claves implicadas. Estos datos serán los que emplearemos para crear las tablas en nuestro SGBD.

2. Para la construcción final del modelo, necesitamos tanteo el recién creado schema.xml como un nuevo fichero build.properties que contendrá la información de la configuración de la BBDD.

```
# Driver de la BBDD
propel.database = pgsql|mysql|sqlite|mssql|oracle
```

```
# Nombre del proyecto
propel.project = <…>
```

Se ha de tener en cuenta que para el correcto funcionamiento, el nombre del proyecto debe ser el mismo que el de la BBDD.

3. Hemos de dejar estos dos ficheros schema.xml y build.properties en el mismo nivel dentro del subdirectorio del proyecto, solo nos queda hacer la llamada a Propel para que nos genere las clases correspondientes, esto es pasándole el parametro “om”, Object Model generador.

```
Propel-gen om
```

Por cada tabla en la BBDD, Propel crea tres clases de PHP:

- Una clase modelo, lo que representa una fila en la base de datos.
- Una clase Peer, ofreciendo constantes estáticas y métodos sobre todo por compatibilidad con versiones anteriores de Propel.
- Una clase Consulta que sirve para trabajar sobre una tabla para poder recuperar y actualizar filas.

4. Propel además también puede generarnos el fichero con el código SQL completo de nuestras tablas, para ello, el parámetro de llamada debe ser “sql”, no es obligatorio para el funcionamiento de Propel pero puede ser interesante para tener el modelo de datos y poder llevarlo a otro SGBD si lo consideramos necesario.

5. Ajustes de Conexión del Runtime. Ahora se debe agregar un archivo de configuración para que las clases de objetos generadas y el runtime de Propel se puedan conectar a la base de datos, y registrar la actividad de Propel. Para ello crearemos un fichero con el nombre runtime-conf.xml que mantendrá el siguiente esqueleto, el ejemplo define una conexión a MySQL:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<config>
  <!—Uncomment this if you have PEAR Log installed
  <log>
    <type>file</type>
    <name>/path/to/propel.log</name>
    <ident>propel-bookstore</ident>
    <level>7</level>
  </log>

  <propel>
    <datasources default=””>
      <datasource id=””>
        <adapter>mysql</adapter>
        <connection>
          <dsn>mysql:host=localhost;dbname=</dsn>
          <user></user>
          <password></password>
        </connection>
      </datasource>
    </datasources>
  </propel>
</config>
```

6. Ahora creamos la configuración para el Runtime mediante la llamada a

```
propel-gen convert-conf.
```

Esto nos genera en el directorio \conf de nuestra aplicación el siguiente fichero

```
<nombre >-conf.php
```

Donde <nombre> es el nombre del proyecto definido en el fichero *build.properties*

CASO 2: BBDD ya creada en un SGBD, mediante ingeniería inversa.

Partimos de un punto mas próximo a la realidad como es la incorporación de un ORM a un entorno ya existente, por lo que evidentemente tenemos también la BBDD creada y funcionando. El proceso es bastante similar, solo difiere algunos puntos.

1. En primer lugar hemos de generar el fichero *build.properties* el cual tendrá el siguiente código dentro

```
propel.project = <nombre>

# The Propel driver to use for generating SQL, etc.
propel.database = <driver>

# This must be a PDO DSN
propel.database.url = <driver>:dbname=<nombre>
propel.database.user = <user>
# propel.database.password =<pwd>
```

2. Una vez tenemos el fichero build.properties se hace la llamada a Propel para que nos genere el fichero XML con la estructura de la BBDD esta vez haremos uso del parámetro “reverse”.

```
propel-gen reverse
```

Con esto obtendremos un fichero *schema.xml* que contendrá la especificación de la BBDD en XML.

3. Ahora como en el caso 1, nos queda hacer la llamada a Propel para que nos genere las clases correspondientes, esto es pasandole el parámetro “om”, Object Model generador.

```
propel-gen om
```

Recordemos que por cada tabla en la BBDD, Propel crea tres clases de PHP:

- una clase modelo.
- una clase Peer.
- una clase Consulta.

4. Llegados a este punto, los pasos a seguir son los mismos que las que tenemos en el caso anterior, a partir del punto 1.d.

Llegados a este punto, ya tenemos la estructura de clases para trabajar con Propel en nuestra aplicación, ahora tendremos que incluir la llamada correspondiente.

Mediante esta llamada se incluye la librería de propel, y además inicializamos las clases particulares para que nuestra aplicación haga uso de ella.

```
// Include the main Propel script
```

```
require_once 'propel/Propel.php';

// Initialize Propel with the runtime configuration
Propel::init("/<proyecto>/build/conf/bookstore-conf.php");

// Add the generated 'classes' directory to the include path
set_include_path("/<proyecto>//build/classes" . PATH_SEPARATOR .
 get_include_path());
```

5.3.2.4. Uso desde gvHidra

Vamos ha realizar un ejemplo.

Disponemos de una tabla, TrecjatReclamante, que tiene la siguiente descripción:

```
CREATE TABLE trecjat_reclamante
(
  nif character varying(15) NOT NULL,
  nombre character varying(25),
  apellido1 character varying(50),
  apellido2 character varying(50),
  direccion character varying(100),
  poblacion character varying(100),
  provincia character varying(100),
  cp character varying(10),
  email character varying(25),
  fax character varying(10),
  telefono character varying(10),
  movil character varying(10),
  nombreempresa character varying(50),
  pais character varying(30) DEFAULT 'España'::character varying,
  CONSTRAINT pk_reclamante PRIMARY KEY (nif)
)
WITH (OIDS=TRUE);
ALTER TABLE trecjat_reclamante OWNER TO recjat;
GRANT ALL ON TABLE trecjat_reclamante TO recjat;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE trecjat_reclamante TO rrecjat;
```

Partiendo de esta descripción de la tabla, Propel genera las siguientes clases (sólo mostramos las mas relevantes):
dbgvhidra/build/classes/dbgvhidra/TrecjatReclamante.php

```
<?php


/**
 * Skeleton subclass for representing a row from the 'trecjat_reclamante'
 table.
 *
 *
 *
```

```
 * You should add additional methods to this class to meet the
 * application requirements.  This class will only be generated as
 * long as it does not already exist in the output directory.
 *
 * @package    propel.generator.dbgvhidra
 */
class TrecjatReclamante extends BaseTrecjatReclamante {

} // TrecjatReclamante
```

dbgvhidra/build/classes/dbgvhidra/TrecjatReclamanteQuery.php

```
<?php


/**
 * Skeleton subclass for performing query and update operations on the
 'trecjat_reclamante' table.
 *
 *
 *
 * You should add additional methods to this class to meet the
 * application requirements.  This class will only be generated as
 * long as it does not already exist in the output directory.
 *
 * @package    propel.generator.dbgvhidra
 */
class TrecjatReclamanteQuery extends BaseTrecjatReclamanteQuery {

} // TrecjatReclamanteQuery
```

dbgvhidra/build/classes/dbgvhidra/om/BaseTrecjatReclamante.php

```
<?php

/**
 * Base class that represents a row from the 'trecjat_reclamante' table.
 *
 *
 *
 * @package    propel.generator.dbgvhidra.om
 */
abstract class BaseTrecjatReclamante extends BaseObject  implements Persistent
{

 /**
  * Peer class name
  */
  const PEER = 'TrecjatReclamantePeer';
```

```
 /**
  * The Peer class.
  * Instance provides a convenient way of calling static methods on a class
  * that calling code may not be able to identify.
  * @var        TrecjatReclamantePeer
  */
 protected static $peer;

 /**
  * The value for the nif field.
  * @var        string
  */
 protected $nif;

 /**
  * The value for the nombre field.
  * @var        string
  */
 protected $nombre;

 /**
  * The value for the apellido1 field.
  * @var        string
  */
 protected $apellido1;

 /**
  * The value for the apellido2 field.
  * @var        string
  */
 protected $apellido2;

 /**
  * The value for the direccion field.
  * @var        string
  */
 protected $direccion;

 /**
  * The value for the poblacion field.
  * @var        string
  */
 protected $poblacion;

...

 /**
  * Flag to prevent endless save loop, if this object is referenced
  * by another object which falls in this transaction.
  * @var        boolean
  */
 protected $alreadyInSave = false;
```

```
 /**
  * Flag to prevent endless validation loop, if this object is referenced
  * by another object which falls in this transaction.
  * @var        boolean
  */
 protected $alreadyInValidation = false;

 /**
  * Applies default values to this object.
  * This method should be called from the object's constructor (or
  * equivalent initialization method).
  * @see        __construct()
  */
 public function applyDefaultValues()
 {
  $this->pais = 'España';
 }

 /**
  * Initializes internal state of BaseTrecjatReclamante object.
  * @see        applyDefaults()
  */
 public function __construct()
 {
  parent::__construct();
  $this->applyDefaultValues();
 }

/*Setters AND Getters*/
...
/*END Setters AND Getters*/

 /**
  * Indicates whether the columns in this object are only set to default
 values.
  *
  * This method can be used in conjunction with isModified() to indicate
 whether an object is both
  * modified _and_ has some values set which are non-default.
  *
  * @return     boolean Whether the columns in this object are only been set
 with default values.
  */
 public function hasOnlyDefaultValues()
 {
   if ($this->pais !== 'España') {
    return false;
   }

  // otherwise, everything was equal, so return TRUE
  return true;
 } // hasOnlyDefaultValues()
```

```
 /**
  * Hydrates (populates) the object variables with values from the database
 resultset.
  *
  * An offset (0-based "start column") is specified so that objects can be
 hydrated
  * with a subset of the columns in the resultset rows.  This is needed, for
 example,
  * for results of JOIN queries where the resultset row includes columns from
 two or
  * more tables.
  *
  * @param      array $row The row returned by PDOStatement-
>fetch(PDO::FETCH_NUM)
  * @param      int $startcol 0-based offset column which indicates which
 restultset column to start with.
  * @param      boolean $rehydrate Whether this object is being re-hydrated
 from the database.
  * @return     int next starting column
  * @throws     PropelException  - Any caught Exception will be rewrapped as a
 PropelException.
  */
 public function hydrate($row, $startcol = 0, $rehydrate = false)
 {
  try {

    $this->nif = ($row[$startcol + 0] !== null) ? (string) $row[$startcol +
 0] : null;
    $this->nombre = ($row[$startcol + 1] !== null) ? (string) $row[$startcol +
 1] : null;
    $this->apellido1 = ($row[$startcol + 2] !== null) ? (string) $row[$startcol
 + 2] : null;
    $this->apellido2 = ($row[$startcol + 3] !== null) ? (string) $row[$startcol
 + 3] : null;
    $this->direccion = ($row[$startcol + 4] !== null) ? (string) $row[$startcol
 + 4] : null;
    $this->poblacion = ($row[$startcol + 5] !== null) ? (string) $row[$startcol
 + 5] : null;
    $this->provincia = ($row[$startcol + 6] !== null) ? (string) $row[$startcol
 + 6] : null;
    $this->cp = ($row[$startcol + 7] !== null) ? (string) $row[$startcol + 7] :
 null;
    $this->email = ($row[$startcol + 8] !== null) ? (string) $row[$startcol +
 8] : null;
    $this->fax = ($row[$startcol + 9] !== null) ? (string) $row[$startcol +
 9] : null;
    $this->telefono = ($row[$startcol + 10] !== null) ? (string) $row[$startcol
 + 10] : null;
    $this->movil = ($row[$startcol + 11] !== null) ? (string) $row[$startcol +
 11] : null;
    $this->nombreempresa = ($row[$startcol + 12] !== null) ? (string)
 $row[$startcol + 12] : null;
```

```
    $this->pais = ($row[$startcol + 13] !== null) ? (string) $row[$startcol +
 13] : null;
    $this->resetModified();

    $this->setNew(false);

    if ($rehydrate) {
     $this->ensureConsistency();
    }

    return $startcol + 14; // 14 = TrecjatReclamantePeer::NUM_COLUMNS -
 TrecjatReclamantePeer::NUM_LAZY_LOAD_COLUMNS).

   } catch (Exception $e) {
    throw new PropelException("Error populating TrecjatReclamante object", $e);
   }
 }

 /**
  * Checks and repairs the internal consistency of the object.
  *
  * This method is executed after an already-instantiated object is re-
hydrated
  * from the database.  It exists to check any foreign keys to make sure that
  * the objects related to the current object are correct based on foreign
 key.
  *
  * You can override this method in the stub class, but you should always
 invoke
  * the base method from the overridden method (i.e.
 parent::ensureConsistency()),
  * in case your model changes.
  *
  * @throws     PropelException
  */
 public function ensureConsistency()
 {

 } // ensureConsistency

 /**
  * Reloads this object from datastore based on primary key and (optionally)
 resets all associated objects.
  *
  * This will only work if the object has been saved and has a valid primary
 key set.
  *
  * @param      boolean $deep (optional) Whether to also de-associated any
 related objects.
  * @param      PropelPDO $con (optional) The PropelPDO connection to use.
  * @return     void
  * @throws     PropelException - if this object is deleted, unsaved or
 doesn't have pk match in db
```

```
 */
public function reload($deep = false, PropelPDO $con = null)
{
 if ($this->isDeleted()) {
  throw new PropelException("Cannot reload a deleted object.");
 }

 if ($this->isNew()) {
  throw new PropelException("Cannot reload an unsaved object.");
 }

 if ($con === null) {
  $con = Propel::getConnection(TrecjatReclamantePeer::DATABASE_NAME,
Propel::CONNECTION_READ);
 }

 // We don't need to alter the object instance pool; we're just modifying
this instance
 // already in the pool.

 $stmt = TrecjatReclamantePeer::doSelectStmt($this->buildPkeyCriteria(),
$con);
 $row = $stmt->fetch(PDO::FETCH_NUM);
 $stmt->closeCursor();
 if (!$row) {
  throw new PropelException('Cannot find matching row in the database to
reload object values.');
 }
 $this->hydrate($row, 0, true); // rehydrate

 if ($deep) {  // also de-associate any related objects?

 } // if (deep)
}

/**
 * Removes this object from datastore and sets delete attribute.
 *
 * @param      PropelPDO $con
 * @return     void
 * @throws     PropelException
 * @see        BaseObject::setDeleted()
 * @see        BaseObject::isDeleted()
 */
public function delete(PropelPDO $con = null)
{
 if ($this->isDeleted()) {
  throw new PropelException("This object has already been deleted.");
 }

 if ($con === null) {
  $con = Propel::getConnection(TrecjatReclamantePeer::DATABASE_NAME,
Propel::CONNECTION_WRITE);
```

```
 }

 $con->beginTransaction();
 try {
  $ret = $this->preDelete($con);
  if ($ret) {
   TrecjatReclamanteQuery::create()
    ->filterByPrimaryKey($this->getPrimaryKey())
    ->delete($con);
   $this->postDelete($con);
   $con->commit();
   $this->setDeleted(true);
  } else {
   $con->commit();
  }
 } catch (PropelException $e) {
  $con->rollBack();
  throw $e;
 }
}

/**
 * Persists this object to the database.
 *
 * If the object is new, it inserts it; otherwise an update is performed.
 * All modified related objects will also be persisted in the doSave()
 * method.  This method wraps all precipitate database operations in a
 * single transaction.
 *
 * @param      PropelPDO $con
 * @return     int The number of rows affected by this insert/update and any
referring fk objects' save() operations.
 * @throws     PropelException
 * @see        doSave()
 */
public function save(PropelPDO $con = null)
{
 if ($this->isDeleted()) {
  throw new PropelException("You cannot save an object that has been
deleted.");
 }

 if ($con === null) {
  $con = Propel::getConnection(TrecjatReclamantePeer::DATABASE_NAME,
Propel::CONNECTION_WRITE);
 }

 $con->beginTransaction();
 $isInsert = $this->isNew();
 try {
  $ret = $this->preSave($con);
  if ($isInsert) {
   $ret = $ret AND $this->preInsert($con);
```

```
   } else {
    $ret = $ret AND $this->preUpdate($con);
   }
   if ($ret) {
    $affectedRows = $this->doSave($con);
    if ($isInsert) {
     $this->postInsert($con);
    } else {
     $this->postUpdate($con);
    }
    $this->postSave($con);
    TrecjatReclamantePeer::addInstanceToPool($this);
   } else {
    $affectedRows = 0;
   }
   $con->commit();
   return $affectedRows;
  } catch (PropelException $e) {
   $con->rollBack();
   throw $e;
  }
 }
...

} // BaseTrecjatReclamante
```

Ahora vamos a ver como enganchar la clase que implementamos en gvHidra, gesReclamante, con la generada por el Propel, TRectjatReclamante:

```
<?php
/* ORM */
// Include the main Propel script
require_once 'C:\wamp\bin\php\php5.2.6\PEAR\propel\Propel.php';

// Initialize Propel with the runtime configuration
Propel::init("C:\wamp\www\dbgvhidra\build\conf\dbgvhidra-conf.php");

// Add the generated 'classes' directory to the include path
set_include_path("C:\wamp\www\dbgvhidra\build\classes" . PATH_SEPARATOR .
 get_include_path());

class gesReclamante extends gvHidraForm
{
    public function __construct()
    {
       parent::__construct();
    }//Fin de constructor

    public function preBuscar($objDatos)
    {
        return 0;
    }
```

```
    public function postBuscar($objDatos)
    {
        $a_reclamantes = TrecjatReclamanteQuery::create()->find();
        $res = array();
        foreach($a_reclamantes as $reclamante)
        {
                $row = array();
                $row['NIF'] = $reclamante->getNif();
                $row['Nombre'] = $reclamante->getNombre();
                $row['Apellido1'] = $reclamante->getApellido1();
                $row['Apellido2'] = $reclamante->getApellido2();
                array_push($res,$row);
        }
        $objDatos->setAllTuplas($res);
        return 0;
    }

    public function preModificar($objDatos)
    {
        $data = $objDatos->getAllTuplas();
        if(count($data[0])>0)
        {
            foreach($data as $row)
            {
                $reclamante = TrecjatReclamanteQuery::create()-
>findPK($row['NIF']);
                $reclamante->setNombre($row['Nombre']);
                $reclamante->setApellido1($row['Apellido1']);
                $reclamante->setApellido2($row['Apellido2']);
                $reclamante->setProvincia($row['Provincia']);
                $reclamante->save();
            }
        }
        return 0;
    }

    public function preBorrar($objDatos)
    {
        $data = $objDatos->getAllTuplas();
        if(count($data[0])>0)
        {
            foreach($data as $row)
                $reclamantes = TrecjatReclamanteQuery::create()-
>findPK($row['NIF']);
                $reclamantes -> delete();
        }

        return 0;
    }

    public function preEditar($objDatos)
    {
```

```
        $this->dataSelected = $objDatos->getAllTuplas();
        return 0;
    }

    public function postEditar($objDatos)
    {
        $data = $this->dataSelected;
        $res = array();
        if(count($data[0])>0)
        {
           foreach($data as $row)
           {
                $reclamante = TrecjatReclamanteQuery::create()-
>findPK($row['NIF']);
                $row = array();
                $row['NIF'] = $reclamante->getNif();
                $row['Nombre'] = $reclamante->getNombre();
                $row['Apellido1'] = $reclamante->getApellido1();
                $row['Apellido2'] = $reclamante->getApellido2();
                $row['Provincia'] = $reclamante->getProvincia();
                array_push($res,$row);
           }
         }
         $objDatos->setAllTuplas($res);
         return 0;
    }

    public function preInsertar($objDatos)
    {
        $data = $objDatos->getAllTuplas();
        if(count($data[0])>0)
        {
            foreach($data as $row)
            {
               $reclamante = new TrecjatReclamante();
               $reclamante->setNif($row['NIF']);
               $reclamante->setNombre($row['Nombre']);
               $reclamante->setApellido1($row['Apellido1']);
               $reclamante->setApellido2($row['Apellido2']);
               $reclamante->setProvincia($row['Provincia']);
               $reclamante->save();
            }
        }
        return 0;
    }
}
?>
```

Capítulo 6. Seguridad

6.1. Autenticación de usuarios

6.1.1. Introducción

gvHIDRA permite integrar las aplicaciones con diferentes sistemas de autenficicación e, incluso, permite crear uno específico para nuestra aplicación. Internamente utiliza PEAR::Auth [http://pear.php.net/manual/en/package.authentication.auth.php], por lo que en algunos puntos es conveniente ver su documentación. A partir de la versión de gvHidra 4.1 se ha integrado el proyecto PEAR Auth dentro del nucleo por lo que ya no se necesitará tener instalado en nuestro servidor el paquete Auth. Para comprender el funcionamiento, conviene distinguir dos partes:

1. Autentificación: validar que el usuario es quien dice ser. El framework proporciona para tal efecto un formulario para introducir las credenciales y unos métodos para la comprobación de su validez.

2. Autorización: carga de la información relativa al usuario / aplicación. Esta información se carga en la sesión, y su estructura puede verse en los ejemplos del método postLogin en igep/include/valida/AuthBasic.php, igep/custom/cit.gva.es/auth/AuthWS.php y en Autenticación imap gva [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/auth_gva.html].

A continuación veremos algunos los mecanismos ya implementados y como se tratan estos puntos en cada uno de ellos.

6.1.1.1. comun

Es el método usado por defecto en el tema 'cit.gva.es' y se trata de un sistema en el que la autenficación se delega en otro sistema. Esto quiere decir que el formulario es externo, y en el framework solo se realiza la parte 2). Esta parte se implementa en comun/modulos/valida.php. En el framework se detecta por los parámetros en el GET pasados desde el formulario externo.

Al pulsar opción de salir se cierra la ventana del navegador (ya que se asume que estamos en una ventana nueva).

6.1.1.2. gvPontis

Es el método de ejemplo usado en el tema 'gvpontis'. Muestra un formulario genérico para el paso 1. Los datos para la conexión son usuario 'invitado' y contraseña '1'. En el paso 2 se carga la información necesaria de forma estática.

Para hacer uso de este método hay que copiar el fichero igep/include/valida/login.php a la raíz del proyecto (se puede renombrar), y acceder a él con el navegador.

Al pulsar la opción de salir se vuelve al formulario de conexión.

6.1.1.3. wscmn (Web Service)

Este método es similar al de **comun**, pero sí incluye formulario, y la comprobación se hace con el web service en wscmn. La parte 2) se inicializa haciendo uso de la información recuperada por el web service, por tanto no requiere acceso a base de datos.

Para hacer uso de este método hay que copiar el fichero igep/custom/cit.gva.es/auth/login_ws.php a la raíz del proyecto (se puede renombrar), y acceder a él con el navegador.

Al pulsar la opción de salir se vuelve al formulario de conexión.

6.1.2. Elección del método de Autenticación

Cuando accedemos a la aplicación sin indicar el fichero de inicio, normalmente el servidor web ejecuta el fichero index.php., donde se hacen una serie de comprobaciones para detectar el método que se va a usar. Hay dos formas de elección:

- explícita: el tipo comun se activa con los parámetros pasados en el GET, y para los otros métodos hay que acceder por http al fichero de login.
- implícita: si no hay elección implícita, se buscan los ficheros 'login*.php' en la raíz y se accede al primero, en orden alfabético.

6.1.3. Crear un nuevo método de Autenticación

A continuación se explican los pasos para crear el nuevo método. También podemos consultar la implementación de los tipos gvPontis y wscmn, ya que se han creado de esta forma. También hay disponible un ejemplo [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/auth_gva.html] usando un servidor imap.

Excepto el fichero de indice (paso 1), los otros pueden situarse en cualquier ubicación, y normalmente vendrá determinado en función de donde necesitemos usarlos (aplicación, custom o framework).

6.1.3.1. Paso 1. Clase Principal

En primer lugar tenemos que crear el fichero php que contiene la clase a usar con PEAR::Auth. Esta clase ha de heredar de 'igep/include/valida/gvhBaseAuth.php'. Los métodos que debemos implementar son:

- fetchData: recibe usuario y contraseña y comprueba su validez
- autenticate: es el método llamado desde el paso 1, y si se aceptan las credenciales, llamamos al método open pasándole la URL de la clase que definimos en paso 2. Consultar los ejemplos para más detalles.

6.1.3.2. Paso 2. <DESCRIPCION>

Creamos el fichero que invoca al método autenticate anterior. Este fichero siempre ha de ir en la raiz del proyecto, y es el punto de entrada a la aplicación usando el navegador.

6.1.3.3. Paso 3. <DESCRIPCION>

Creamos la clase validacion, con el método valida (llamado desde el framework), que lo que hace es cargar la información necesaria en la sesión. Podemos llamar al método checkData (de la clase principal) que comprueba si en la sesión está toda la información que requiere gvHIDRA.

6.2. Modulos y Roles

6.2.1. Introducción

Los módulos y roles es la forma usada para controlar el acceso a las distintas partes de un aplicación. No hay que confundir estos módulos con los usados a nivel de los menús para agrupar funcionalidades.

6.2.1.1. Módulos

Los módulos representan permisos que un usuario tiene en una aplicación determinada. Los módulos se asignan cuando se lleva a cabo una asociación entre un usuario y una aplicación, y se cargan en el inicio de la aplicación. Cada módulo tiene un código, una descripción y opcionalmente puede tener un valor. Cada usuario puede tener asignado uno o varios módulos, y para cada uno puede modificar su valor. Algunos usos habituales de módulos son el controlar si un usuario puede acceder a una opción de menú, o si puede ver/editar un campo determinado de una pantalla, o para guardar alguna información necesaria para la aplicación (departamento del usuario, año de la contabilidad, ...). También suelen usarse para definir variables globales para todos los usuarios, aunque en este caso es recomendable asignarlos a roles (ver apartado siguiente).

Ejemplo:

- Todos los usuarios que consulten la aplicación PRESUPUESTARIO deben tener el módulo M_CONSUL_PRESUPUESTARIO, es decir, nadie que no tenga ese módulo asociado podrá acceder al apartado de listados de la aplicación
- Sólo el personal de la oficina presupuestaria tendrá acceso a la manipulación de datos, es decir el módulo M_MANT_PRESUPUESTARIO
- Sólo técnicos y el jefe de la oficina presupuestaria tendrán tendrán acceso a la entrada de menú "control de crédito". Pues serán aquellos usuarios que tengan el módulo M_MANT_PRESUPUESTARIO, con el valor TECNICO o el valor JEFE.
- Usuarios:
  - Sandra (Administrativa de otro departamento que consulta la aplicación): Tiene el módulo M_CONSUL_PRESUPUESTARIO
  - Juan (Administrativo): Tiene el módulo M_MANT_PRESUPUESTARIO, (bien sin valor, o con el valor PERFIL_ADMD)
  - Pepe (Técnico): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO
  - Pilar (Jefa de la oficina presupuestaria): Tiene el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_JEFE
- Módulos:
  - Nombre: M_CONSUL_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las consultas de la aplicación de presupuestario
    - Valores posibles: <COMPLETAR>
  - Nombre: M_MANT_PRESUPUESTARIO
    - Descripción: Módulos de acceso a las opciones de mantenimiento de la aplicación de presupuestario
    - Valores posibles: PERFIL_ADMD, PERFIL_TECNICO o PERFIL_JEFE

6.2.1.2. Roles

Los roles representan el papel que el usuario desempeña en una aplicación y a diferencia de los módulos, cada usuario sólo puede tener uno y no tienen valor. ¿Entonces para que queremos los roles si podemos

hacer lo mismo usando módulos sin valor? Podemos ver los módulos como los permisos básicos, y los roles como agrupaciones de módulos. Realmente los roles se utilizan para facilitar la gestión de usuarios. Si sólo usamos módulos y por ejemplo tuviéramos 30 módulos, seria muy difícil clasificar a los usuarios ya que seguramente cada uno tendría una combinación distinta de módulos, y cada vez que hubiera que dar de alta un usuario tendríamos que tener un criterio claro para saber cuales de los módulos asignar. Con roles es más simple, ya que es sencillo ver los permisos de cualquier usuario (viendo el role suele ser suficiente), y para añadir nuevos usuarios normalmente solo necesitaremos saber su role. Para que esto sea fácil de gestionar, tendríamos que tener algún mecanismo que nos permitiera asignar módulos a roles, y que los usuarios con un role "heredaran" estos módulos. Lo más flexible seria tener esta información en tablas aunque también se podría empezar haciéndolo directamente en PHP (o ver abajo módulos dinámicos):

```
if ($role == 'admin') {
   // combinacion 1
   $modulos[] = array('borrarTodo'=>array('descrip'=>'borra todo',));
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion11'=>array('descrip'=>'opcion 11',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   $modulos[] = array('opcion13'=>array('descrip'=>'opcion 13',));
   ...
} elseif ($role == 'gestor') {
   // combinacion 2
   $modulos[] = array('verTodo'=>array('descrip'=>'ver todo',));
   $modulos[] = array('opcion12'=>array('descrip'=>'opcion 12',));
   ...
} elseif ($role == '...') {
...
```

De esta forma, añadir un nuevo usuario de tipo administrador consistiría simplemente en indicar su role, en vez de tener que asignarle los N módulos que tienen los administradores.

La solución más flexible seria usar sólo módulos para controlar cualquier característica que pueda ser configurable por usuario, y asignar los módulos a los roles de forma centralizada (bien en base de datos o en el inicio de la aplicación). De esta forma el programador solo tendría que tratar con los módulos, y el analista con los módulos de cada role.

El ejemplo anterior usando roles podría ser:

- módulos: M_CONSUL_PRESUPUESTARIO, M_MANT_PRESUPUESTARIO (ambos sin valor)
- roles:
  - PERFIL_ADMD (módulo M_MANT_PRESUPUESTARIO), asignado a Juan
  - PERFIL_TECNICO (módulo M_MANT_PRESUPUESTARIO), asignado a Pepe
  - PERFIL_JEFE (módulo M_MANT_PRESUPUESTARIO), asignado a Pilar
  - PERFIL_OTROS (módulo M_CONSUL_PRESUPUESTARIO), asignado a Sandra

En este caso las dos soluciones tienen una complejidad similar, aunque las diferencias serán más evidentes conforme aumenten el número de módulos y usuarios. Si por ejemplo decidiéramos que ahora todos los técnicos han de tener un nuevo módulo X, sin usar roles tendríamos que añadir ese módulo a todos los usuarios que tuvieran el módulo M_MANT_PRESUPUESTARIO con valor PERFIL_TECNICO; usando roles seria simplemente añadir el módulo X al role PERFIL_TECNICO.

6.2.2. Uso en el framework

El primer ejemplo de uso de módulos puede encontrarse en la creación de los ficheros xml que da lugar a la pantalla de inicio de la aplicación [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/menu.html], en dichos ficheros se utiliza la etiqueta "controlAcceso" para indicar que módulos necesita tener asignados un usuario para acceder a la opción. Por ejemplo:

```
<opcion titulo="Prueba del árbol" descripcion="Pues eso... prueba del árbol"
 urlAbs="phrame.php?action=IgepPruebaArbol__iniciarVentana"
 imagen="menu/24.gif">
   <controlAcceso><moduloPermitido id="M_INTRANET"/></controlAcceso>
</opcion>
```

Con el párrafo anterior de XML, se indica que la entrada de la aplicación "Prueba del árbol" sólo estará disponible para aquellos usuarios que cuenten entre sus módulos el M_INTRANET. También se puede hacer el control usando un role.

Los módulos podemos usarlos en otros sitios, y podemos acceder a ellos a través de

Tabla 6.1. Tabla de metodos 1

método	descripción
IgepSession::hayModulo(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo.
IgepSession::dameModulo(<modulo>)	Información del módulo.
ComunSession::dameModulos()	Todos los módulos (estáticos) asignados al usuario (se recomienda usar el método con el mismo nombre de IgepSession).

Cuando queremos usar módulos que no estén permanentemente asignados a un usuario, sino que la asignación dependa de otras circunstancias que puedan ir cambiando durante la ejecución de la aplicación, la asignación no la haremos en el inicio de la aplicación. Para poder añadir o eliminar módulos en tiempo de ejecución, y conseguir, entre otras cosas el poder hacer accesibles u ocultar opciones de menú dinámicamente, podemos además usar:

Tabla 6.2. Tabla de metodos 2

método	descripción
IgepSession::anyadeModuloValor(<modulo>, <valor>=null, <descripcion>=null)	Añade un nuevo módulo al usuario
IgepSession::quitaModuloValor(<modulo>, <valor>=null)	Quita el módulo al usuario; si se le pasa valor, sólo lo quita si el valor del módulo coincide con el valor pasado
IgepSession::hayModuloDinamico(<modulo>)	Devuelve un booleano indicando si el usuario tiene asignado el módulo dinámico
IgepSession::dameModuloDinamico(<modulo>)	Información del módulo dinámico
IgepSession::dameModulosDinamicos()	Todos los módulos dinámicos asignados al usuario
IgepSession::dameModulos()	Todos los módulos asignados al usuario

Estos módulos les llamaremos módulos dinámicos. A efectos del framework, no hay diferencia entre los módulos cargados inicialmente y los módulos dinámicos. El plugin CWPantallaEntrada ya incluye dicha

funcionalidad, por lo que si en alguno de los ficheros XML [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/menu.html] aparece una opción como esta:

```
<opcion titulo="Prueba de módulo dinamico"
 descripcion="Ejemplo de activación y descativación de opciones en ejecución"
 urlAbs="http://www.gvpontis.gva.es" imagen="menu/24.gif">
   <controlAcceso><moduloPermitido id="MD_GVA"/></controlAcceso>
</opcion>
```

hasta que en algún punto de nuestro código PHP no realicemos una llamada como esta:

```
IgepSession::anyadeModuloValor("MD_GVA")
```

la opción permanecerá oculta.

Si después de habilitarla queremos volver a ocultarla, basta con ejecutar la opción:

```
IgepSession::quitaModuloValor("MD_GVA")
```

Podemos obtener el role del usuario con el método IgepSession::dameRol().

6.2.2.1. Restricciones en los menús

Para que los cambios sean efectivos de forma visual en los menús, es necesario que se reinterprete el código XML, cosa que sólo se hace cuando se pasa por la pantalla principal de la aplicación, mientras tanto NO será actualizada la visibilidad/invisibilidad de las distintas ramas, módulos y/o opciones. Por tanto, convendrá también añadir en las acciones que correspondan al menú, comprobación explícita de que se tienen los módulos/roles necesarios para el acceso.

6.3. Permisos

<PENDIENTE>

Parte IV. Conceptos Avanzados

Tabla de contenidos

7. Conceptos Avanzados 188
7.1. Excepciones 188
7.1.1. gvHidraSQLException 188
7.1.2. gvHidraLockException 188
7.1.3. gvHidraPrepareException 188
7.1.4. gvHidraExecuteException 188
7.1.5. gvHidraFetchException 189
7.1.6. gvHidraNotInTransException 189
7.2. Log de Eventos 189
7.2.1. Introducción 189
7.2.2. Crear eventos en el log 190
7.2.3. Consulta del Log 191
7.3. Depurando mi aplicación 191
7.4. Envio de correo desde mi aplicación: IgepCorreo 191
7.4.1. Configuración 191
7.4.2. Métodos básicos 191
7.5. Creación de un custom propio para una aplicación de gvHIDRA 192
7.5.1. Pasos previos 192
7.5.2. Archivos CSS 193
7.5.3. Clases más usadas de bootstrap 193
7.5.4. Iconos mediante fuentes tipográficas (icon-fonts) 194
7.5.5. Tipos de clases 194
7.5.6. Cosas a tener en cuenta 195
7.5.7. Estilos generales de la aplicación 195
7.5.8. Personalización de Logos 196
7.5.9. Panel login 198
7.5.10. Pantalla de inicio 202
7.5.11. Acerca de 205
7.5.12. Ventanas de aviso 208
7.5.13. Partes genéricas de los paneles FIL LIS EDI 211
7.5.14. Partes genéricas de los paneles LIS EDI 217
7.5.15. Panel FIL 219
7.5.16. Panel EDI 221
7.5.17. Panel LIS 223
7.5.18. Panel maestro-detalle 226
7.5.19. Panel árbol 227
7.5.20. Solapas 229
7.5.21. Debugger 231
7.5.22. Ejemplos prácticos 232
8. Bitacora de cambios aplicados a gvHidra 240
8.1. Como migrar mis aplicaciones a otra versión de gvHidra 240
8.1.1. Versión 4.4.22 240
8.1.2. Versión 4.4.13 240
8.1.3. Versión 4.2.19 242
8.1.4. Versión 4.2.0 242
8.1.5. Versión 4.0.x 245
8.1.6. Versión 4.0.0 246
8.1.7. Versión 3.1.0 248

Capítulo 7. Conceptos Avanzados

7.1. Excepciones

Con el objetivo de hacer la gestión de errores más flexible para el programador, existen algunas funcionalidades de gvHIDRA que usan excepciones. Para ello se ha creado una jerarquia de excepciones que irá creciendo según las necesidades. Esta jerarquía de excepciones es la siguiente:

Tabla 7.1. Tabla de Excepciones

excepción	descripción
Exception	Excepción definida en PHP
gvHidraException	Excepción usada como base de todas las excepciones del framework
gvHidraSQLException	cCase base para excepciones relacionadas con SQL
gvHidraLockException	Excepción producida cuando no se puede bloquear un recurso. Ver sección de transacciones en Conexiones a SGBD .
gvHidraPrepareException	En sentencias preparadas (ver sección de Conexiones a SGBD), cuando no se pu preparar una sentencia
gvHidraExecuteException	En sentencias preparadas, cuando no se puede ejecutar una sentencia
gvHidraFetchException	En sentencias preparadas, cuando no se puede recuperar datos
gvHidraNotInTransException	Excepción producida cuando en una operación se requiere transacción en curso.

A continuación vamos a explicar algunos métodos disponibles.

7.1.1. gvHidraSQLException

Define una propiedad para almacenar el objeto error del PEAR. El objeto se asigna en el constructor, y se puede recuperar con un método.

Métodos:

- **__construct($message='', $code=0, $prev_excep=null, $pear_err=null)**

  Posibilidad de asignar objeto error. El tercer parámetro sólo tiene efecto a partir de PHP 5.3.

- **getSqlerror()**

  Obtener el objeto error.

7.1.2. gvHidraLockException

7.1.3. gvHidraPrepareException

7.1.4. gvHidraExecuteException

7.1.5. gvHidraFetchException

7.1.6. gvHidraNotInTransException

7.2. Log de Eventos

Funcionamiento del log y los diferentes usos que se le puede dar.

Ponemos especial énfasis en su utilización como método de Debug del código en desarrollo.

7.2.1. Introducción

El objetivo de este módulo es registrar ciertos eventos, acciones, movimientos, operaciones, ... para poder averiguar lo que está haciendo la aplicación. Por defecto éstos se registran en una tabla de sistema. Esta tabla se llama tcmn_errlog y hay que crearla previamente a usarla. A continuación mostramos el script de dicha tabla para PostgreSQL:

```
CREATE TABLE tcmn_errlog
(
  iderror    numeric(9) NOT NULL,
  aplicacion varchar(10) NOT NULL,
  modulo     varchar(10),
  version    varchar(10) NOT NULL,
  usuario    varchar(30) NOT NULL,
  fecha      timestamp NOT NULL,
  tipo       numeric(2) NOT NULL,
  mensaje    text NOT NULL,
  CONSTRAINT tcmn_errlog_pkey PRIMARY KEY (iderror)
);
grant select, insert on tcmn_errlog to <user_web>;

// Para postgresql y oracle hay que crear la secuencia:
CREATE SEQUENCE scmn_id_errlog;
grant select on scmn_id_errlog to <user_web>;

// Para mysql añadir a iderror auto_increment

// Para oracle: cambiar varchar por varchar2, timestamp por timestamp(0) y
 crear sinonimos:
create public synonym tcmn_errlog for <owner>.tcmn_errlog;
create public synonym scmn_id_errlog for <owner>.scmn_id_errlog;
```

Más adelante se pueden definir otros métodos como podría ser el correo, ficheros, ... o mejor Pear::Log [http://pear.php.net/package/Log]. En los ficheros de configuración [http://zope.coput.gva.es/proyectos/igep/trabajo/igep/configphp.html] (propiedad DSNZone/dbDSN con id 'gvh_dsn_log') se puede definir una fuente de datos válida para añadir registros a la tabla.

Esto habitualmente lo usaremos para detectar errores de una aplicación en explotación o para poder realizar una traza detallada de una acción en desarrollo.

Hemos realizado una clasificación de los eventos que se registran en la tabla dependiendo de su gravedad:

Tabla 7.2. Tabla de clasificación de los eventos

tipo	descripción
PANIC	Errores graves, la aplicación no deberia seguir ejecutándose
ERROR	Errores
WARNING	Errores menores
NOTICE	Información a nivel de Auditoría
DEBUG_USER	Mensajes de traza (debug) del programador
DEBUG_IGEP	Mensajes de traza (debug) de gvHidra

Por defecto se registran todos los eventos de los dos primeros tipos. Si queremos cambiarlo, podemos hacerlo indicando a la aplicación el nivel de sensibilidad. Es decir, indicar que tipo de eventos se tienen que registrar. Para ello se puede escoger de los siguientes valores:

- **LOG_NONE** : No registra nada.
- **LOG_ERRORS**: Registra únicamente ERROR y PANIC. (valor por defecto)
- **LOG_AUDIT**: Registra todos los anteriores más WARNING y NOTICE.
- **LOG_ALL**: Registra todas las acciones.

El valor lo podemos poner de forma estática en el atributo logSettings del fichero gvHidraConfig.inc.xml de la aplicación o de forma dinámica con el método ConfigFramework->setLogStatus (ver configuración).

Hay que tener precaución con esta variable, para que en explotación no se genere más información que la necesaria. Normalmente le asignaremos un valor u otro en función de si estamos en producción o no.

Otra de las precauciones es el control del acceso a la consola del Debug. Con el parámetro security podemos indicar que Rol o Módulo debe tener el usuario para poder acceder a la consola. Automáticamente, se realizará este control al cargar la consola.

7.2.2. Crear eventos en el log

Un programador puede crear eventos en el log simplemente llamando al método setDebug de la clase IgepDebug.

Ejemplo:

```
function creacionInformes(){
  IgepDebug::setDebug(DEBUG_USER,'Pasamos a crear los informes');
  ....
}
```

En este ejemplo, y dependiendo del valor del atributo logSettings, el programador inserta una entrada en el log que indica que se ha pasado por el método creacionInformes.

El programador podrá hacer uso del log indicando la gravedad del mensaje que desee atendiendo a la clasificación. Se recomienda hacer uso de DEBUG_USER si se esta realizando un debug en desarrollo y WARNING o NOTICE si se quiere realizar auditoría de lo que realicen los usuarios en explotación.

7.2.3. Consulta del Log

Para ver los eventos generados, se ha creado una consola que permite consultar lo que se inserta en la tabla de sistema. Se recomienda a los programadores que hagan uso de ella añadiendo en el fichero menuHerramientas.xml la siguiente linea:

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación" urlAbs="igep/
_debugger.php" imagen="menu/51.gif"/>
```

Hay que tener precaución en este último punto para que esta opción no esté accesible para los usuarios finales.

7.3. Depurando mi aplicación

<EJEMPLO PRÁCTICO DE COMO DEPURAR UNA APP>

7.4. Envio de correo desde mi aplicación: Igep-Correo

7.4.1. Configuración

El framework proporciona una clase para facilitar el proceso de envios de correos electrónicos. Esta clase, IgepCorreo, hace uso del paquete PEAR::Mail por lo que antes de utilizarla, deberemos comprobar que dicho paquete está instalado en nuestro servidor. Precisamente porque no todo el mundo necesita envio de correo y, no todo el mundo instala el paquete PEAR:Mail, esta clase no viene incluida por defecto con el framework. Para hacerlo tendremos que incluir en nuestro fichero include:

```
include_once "igep/include/IgepCorreo.php";
```

Finalmente, para poder configurar la clase, debemos indicarle el servidor de correo que vamos a utilizar. Desde la versión 4.0.0, gvHidra permite configurar dicho servidor a través de los ficheros de configuración estática del custom y/o el framework (gvHidraConfig.inc.xml). A continuación, mostramos un ejemplo de configuración:

```
 <smtpServer>
   <smtpHost>smtp.gva.es</smtpHost>
   <smtpPort>25</smtpPort>
   <smtpUser>usuario</smtpUser>
   <smtpPassword>password</smtpPassword>
 </smtpServer>
```

Si no se añade ningún servidor, por compatibilidad hacia atrás, el servidor por defecto será el de la Generalitat Valenciana.

7.4.2. Métodos básicos

La clase IgepCorreo, es una clase estática; por lo que son los métodos públicos básicos los que nos ofreceran la funcionalidad. Básicamente, esta funcionalidad es el envío de correo sin anexo y el envío de correo con anexo. Cabe destacar que para ambos casos, el framework encola los mensajes para evitar saturar el servidor de correo.

En ambos métodos, cuando hay una lista de destinatarios, el mensaje se envía individualmente a cada uno, por lo que cada destinatario no conocerá el resto de destinatarios. Devuelve falso si falla el envio a alguno de los destinatarios, o cierto en otro caso.

7.4.2.1. sinAnexo

Parámetros:

- $from
- $to: array con los destinatarios del correo
- $subject: asunto
- $msg: cuerpo del mensaje
- $responder_a

7.4.2.2. conAnexo

Parámetros (además de los del método sinAnexo):

- $tmp_fich
- $tipo_fich
- $nom_fich

Estos parámetros son los necesarios para enviar un fichero como anexo. El método sólo está preparado para enviar un anexo.

7.5. Creación de un custom propio para una aplicación de gvHIDRA

El propósito del presente manual es servir como una guía para que, partiendo de un custom de aplicación básico (el custom "default" incluido en la distribución), podamos modificar/añadir características propias para nuestra aplicación .

El manual no explica sintaxis u opciones del lenguaje de hojas de estilo css, para ello ya hay varios tutoriales en la web que se pueden consultar par un conocimiento mas amplio de css.

Lo que se especificará a lo largo del manual será, la correspondencia entre las diferentes partes de una ventana de una aplicación gvHIDRA (barra superior, inferior, paneles, botones, etc), y los clases o secciones dentro del archivo .css responsables de controlar las características y apariencias de dichas partes .

7.5.1. Pasos previos

Para no empezar desde cero y ahorrarnos tiempo de teclear muchas directivas, primero copiamos un custom base que esta incluido en la distribución (el directorio "default" que esta en /ruta-de-nuestro-proyecto/igep/custom/) y lo pegamos en la misma ruta (cambiándole de nombre al que queremos que sea el nombre de nuestro custom) .

Una vez hecho eso modificamos las rutas dentro de los siguientes archivos para que

- /ruta-de-nuestro-proyecto/igep/custom/nuestroCustom/**include.php**
- /ruta-de-nuestro-proyecto/igep/custom/nuestroCustom/**include_class.php**

y donde haya una referencia a "*default*" la remplazamos por el nombre que le hemos dado al directorio de nuestro custom .

Por ultimo modificamos la directiva de configuración **<customDirName>** en el archivo /ruta-de-nuestro-proyecto/igep/**gvHidraConfig.inc.xml** indicando el nombre de nuestro directorio de custom (solamente el nombre, no la ruta completa):

```
<customDirName>nombre-dir-custom</customDirName>
```

y además especificamos el nombre de nuestra aplicación en el fichero /ruta-de-nuestro-proyecto/**gvHidraConfig.inc.xml**:

```
<applicationName>nombre_app</applicationName>
```

7.5.2. Archivos CSS

Vamos a describir la correspondencia entre las partes de la ventana con cada una de los selectores que se definen en los ficheros css del custom.

En la siguiente ruta de nuestro custom (/igep/custom/<nuestro custom>/css) nos encontraremos los ficheros css necesarios para definir el aspecto de la aplicación. Encontraremos los siguientes ficheros:

- **aplication-login.css**

  En él se define los estilos de la pantalla de validación de la aplicación.

- **apl_ALL.css**

  En él se define los estilos CSS principales de la aplicación.

- **breadcrumb.css**

  En él se definen los estilos para las "migas de pan" que aparecerá, si se activan, en la pantalla principal.

- **bumpbox.css**

  En él se define el estilo del bumpbox para las imágenes que lo utilicen.

- **debugger.css**

  En él se define el estilo de la consola de Log o Debug de la aplicación.

- **font-awesome.css**

  Estilos para cargar la librería de fuentes FontAwesome.

- **layersmenu-cit.css**

  Estilos que se aplican en el menú desplegable de la barra superior.

- **alertify.core.css y alertify.default.css**

  Estilos que se aplican en los mensajes de aviso, error...

7.5.3. Clases más usadas de bootstrap

- **Row**

  Se usa como contenedor.

- **Col-xs-X col-sm-X col-md-X**

  Se usa para dividir un contenedor en columnas a diferentes resoluciones

- **Text-left**

  Colocar elementos a la izquierda.

- **Text-center**

  Coloca elementos en el centro.

- **Text-right**

  Coloca elementos a la derecha

- **Table**

  Proporciona estilos generales propios de una tabla.

- **Table-responsive**

  Adapta la tabla en diferentes resoluciones insertando un scroll inferior.

7.5.4. Iconos mediante fuentes tipográficas (icon-fonts)

Los iconos por fuentes se implementan por medio de una tipografía especial, en la que en lugar de caracteres que representan letras hay caracteres que representan imágenes. Con las ventajas que ello conlleva (evitar solicitudes al servidor, adaptación más "responsive", personalizarlos con css...)

gvHIDRA incluye la librería **Glyphicons** (http://glyphicons.com), con ella se crean los iconos internos del framework (p.ej. paginador). Todo lo referente a los estilos de esta libreria se encuentra en el fichero apl_ALL.css del custom.

El juego de iconos de la librería Glyphicons puede que no cubra algún icono en concreto, por lo que se necesite añadir una nueva librería. Por esto se ha incluido también la librería **Font-Awesome** (http://fortawesome.github.io/Font-Awesome/) que amplía el juego de iconos.

Es posible que surja la necesidad de incluir alguna otra librería, no hay problema. Una vez descargada la librería que interese, hay que copiarla en el directorio **custom/fonts** y el fichero **font-awesome.css** copiarlo en el directorio **custom/css**. Una vez están los ficheros en su sitio, se debe hacer un import de la css de la fuente en el fichero **apl_ALL.css** y cargar la fuente correspondiente con **@font-face**.

La ventaja de que sean iconos por fuentes es que permite que mediante CSS se puedan personalizar (color, sombras, grosor...)

```
.glyphicon-warning-sign {
 color: #fff;
}
```

7.5.5. Tipos de clases

En la aplicación podemos encontrarnos tres tipos de clases.

- **Clase de la aplicación**

Clases propias de la aplicación donde a lo largo del documentos las llamaremos clase de la aplicación.

- **Clase de Bootstrap**

  Clases propias de Bootstrap sin modificar, a estas las llamaremos clase de Bootstrap.

- **Clase híbrida**

  Las terceras son clases propias de Bootstrap pero que se modifican en nuestras hojas de estilos para bien reemplazar estilos que vienen predeterminados o bien para incluir nuevos estilos sobre estas clases, a este tipo de clases las llamaremos clase híbrida.

7.5.6. Cosas a tener en cuenta

- **Clases con ***

  En las capturas de pantalla donde se muestran los bloques a lo largo de la guía, cada bloque estará señalizado con una capa de un color, por realizar el proceso de una manera clara en algunos contenedores que ocupen todo el bloque se señalizará con una capa delgada indicando como información a seguir el ancho de esa capa para visualizar lo que ocupa ese contenedor.

  Como por ejemplo en la siguiente captura de pantalla:

  Donde el punto 1 y el punto 2 llevan un asterisco, lo que significa que el contenedor al que pertenecen dichos puntos tienen el ancho que se muestra en la barra de color asignada a su clase dejando únicamente la altura al sentido común del diseñador.

- **Clases principales**

  Cada contenedor estará señalizado con su clase principal para distinguirlos unos de otros.

- **Nuevos estilos**

  Algunos elementos usan clases de las hojas de estilos de Bootstrap, si se quiere modificar o incluir nuevos estilos a dicha clase la forma correcta de trabajar es incluir esa misma clase en nuestro custom y desde la hojas de estilo propias de la aplicación realizar los cambios que se deseen.

7.5.7. Estilos generales de la aplicación

Estos son estilos que se usan de manera global en la mayor parte de la aplicación y están definidos en la hoja de estilos aplicacion.css.

- **Etiqueta de la aplicación: body**

  Estos estilos se incluyen sobre la etiqueta body.

  Los siguientes estilos añaden una textura sobre el fondo de la aplicación.

```
.body {
        background-image: url("../images/subtle_dots.png");
        background-position: center center;
        background-attachment: fixed;
```

```
       }
```

- **Clase de la aplicación: text**

  En esta clase se configura la mayor parte del texto que se muestra en la aplicación.

```
.text {
     border: 0 none;
     color: #55585d;
     font-family: Open Sans,Roboto,Helvetica,Arial;
     font-size: 13px;
     font-weight: normal;
    }
```

- **Clase híbrida: row**

  Se ha modificado la clase inicial de Bootstrap.

  Esta clase de Bootstrap engloba a los contenedores (filas).

```
.row {
       margin-left: 0px;
       margin-right: 0px;
      }
```

7.5.8. Personalización de Logos

En versiones anteriores, los LOGOS (pantalla principal, acerca de...) se personalizaban cambiando en el directorio CUSTOM determinadas imágenes. Desde la versión v.4.4.30, la personalización se realiza vía CSS3, siguiendo el habitual sistema de sobrecarga GVHIDRA -> CUSTOM -> Aplicación.

En un primer nivel, GVHIDRA (el FW) fija una imagen de logo por defecto (fichero **aplicacion/igep/igep.css**).

```
.gvHAboutLogo,
.gvHFooterLogo {
    background-repeat: no-repeat;
    background-color: transparent;
    background-size: 100%;
    background-position: top, left;
    content: "";
    display: none;
}

.gvHAboutLogo {
    margin: 1em auto;
}
.gvHFooterLogo {
    margin-left: 1em;
    margin-right: 1em;
    vertical-align: middle;
}

#gvHFooterLogo_1 {
    background-image: url("images/logo_gvhidra.gif");
```

```
    display: inline-block;
    width: 225px
    height: 159px;
}

#gvHAboutLogo_1 {
    background-image: url("images/logo_gvhidra.gif");
    display: block;
    width: 112px;
    height: 79px;
}
```

Posteriormente, esas clases CSS pueden ser sobrecargadas por el CUSTOM en **aplicacion/custom/greyStyle/apl_ALL.css**.

```
/* Propidades de la capa LOGO  */
#gvHFooterLogo_1 {
    background-image: url("../images/logos/logo.gif");
    display: inline-block;
    width: 241px;
    height: 58px;
}

/* Propidades de la capa LOGO en el Acerca de... */
#gvHAboutLogo_1 {
    background-image: url("../images/logos/logo.gif");
    display: block;
    width: 241px;
    height: 58px;
}
```

Finalmente, puede añadirse en el fichero CSS de la aplicación (**aplicacion/css/AppStyle.css**).

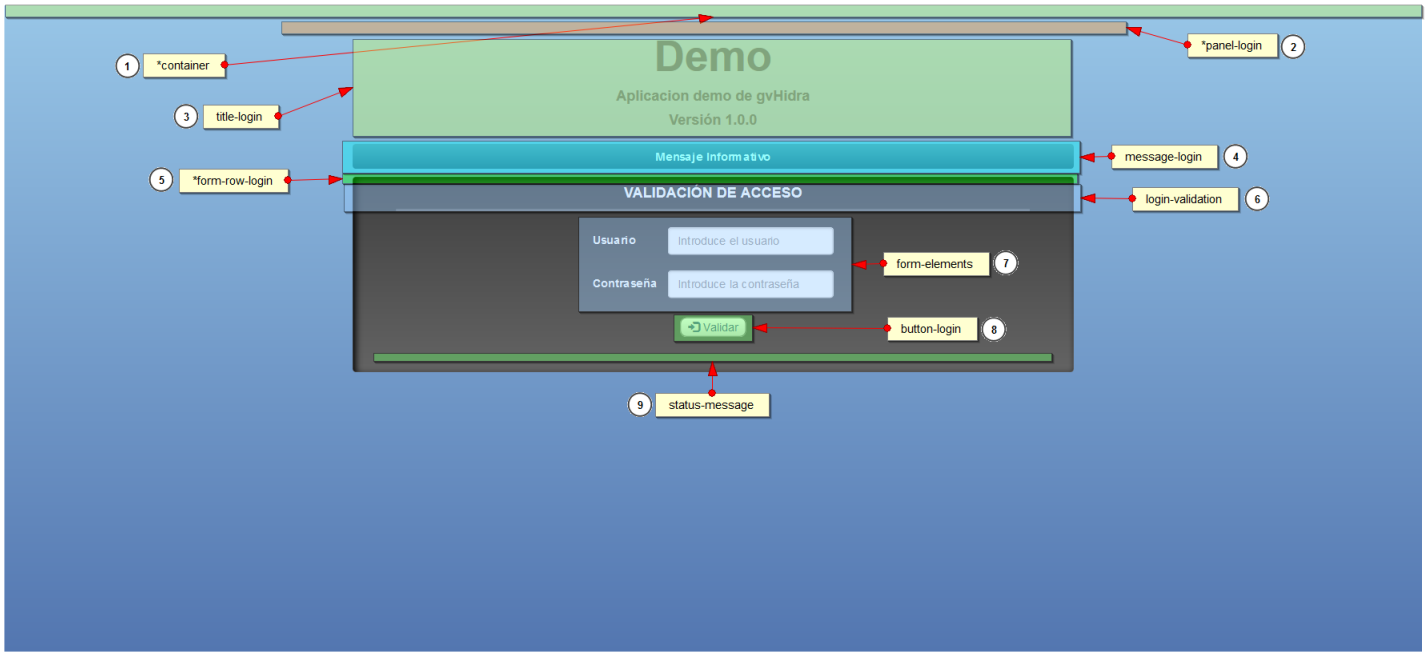
```
#gvHFooterLogo_1 {
    display: inline-block;
    width: 200px;
    height: 200px;
    background-image: url("../images/miLogodeApp.png");
    opacity: .75;
    filter: alpha(opacity=75);
}

#gvHFooterLogo_2 {
    display: inline-block;
    width: 200px;
    height: 200px;
    background-image: url("../images/miLogodeApp2.png");
    opacity: .75;
    filter: alpha(opacity=75);
}

#gvHAboutLogo_1 {
```

```
    display: block;
    width: 100px;
    height: 100px;
    background-image: url("../images/miLogodeApp1.png");
    opacity: 1;
    filter: alpha(opacity=100);
}

#gvHAboutLogo_2 {
    display: block;
    width: 90px;
    height: 90px;
    background-image: url("../images/miLogodeApp2.png");
    opacity: 1;
    filter: alpha(opacity=100);
}

#gvHAboutLogo_3 {
    display: block;
    width: 90px;
    height: 90px;
    background-image: url("../images/miLogodeApp3.png");
    opacity: 1;
    filter: alpha(opacity=100);
}
```

En el ejemplo anterior, veríamos dos imágenes (logos) en la pantalla principal, y tres en la opción de menú **Acerca de...**.

7.5.9. Panel login

Los estilos que usa el panel de login se encuentra en el archivo aplication-login.css



- **Punto 1**

  Esta clase es la que proporciona el contenedor más grande donde contendrá los demás contenedores de la página.

  Clase híbrida: container

  La clase container es propia de la librería de estilos Bootstrap y contiene unos estilos ya definidos de forma predeterminada, en la hoja de estilos aplication-login.css modificamos o reemplazamos algunos atributos que vienen de forma predeterminada.

```
.container {
    background:linear-gradient(#98c6e7,#5376b0);
    background:-o-linear-gradient(#98c6e7,#5376b0);
    background:-moz-linear-gradient(#98c6e7,#5376b0);
    background:linear-gradient(#98c6e7,#5376b0);
    padding-top:15px;
    height:100%;
    width: 100%;
   }
```

- **Punto 2**

  Contenedor del panel de login.

  Clase de la aplicación: panel-login

```
.panel-login  {
    width: 50%;
    margin: 0 auto;
    }
```

- **Punto 3**

  Contenedor que contiene la información del título de la aplicación, la descripción y la versión.

  Clase de la aplicación: title-login

  Clase de BootStrap: text-center

```
.title-login {
     color:#4e688d;
   }

.title-login h2 {
    font-size: 53px;
    font-weight: bold;
   }

.title-login h4 {
     font-weight: bold;
   }
```

- **Punto 4**

  Esta clase contiene la información donde aparece el mensaje informativo.

Clase de la aplicación: message-login

Clase de BootStrap: text-center

```
.message-login  {
    background:linear-gradient(#6f91ac,#476286);
    background:-o-linear-gradient(#6f91ac,#476286);
    background:-moz-linear-gradient(#6f91ac,#476286);
    background:linear-gradient(#6f91ac,#476286);
    box-shadow: 2px 2px c5px #bbbbbb inset;
    border-radius: 4px;
    color:#fff;
    padding:8px;
    font-weight: bold;
   }
```

- **Punto 5**

  Esta clase contiene el contenedor del formulario.

  Clase de la aplicación: form-row-login

  Clase de BootStrap: text-center

```
.form-row-login {
    background:linear-gradient(#404040,#636363);
    background:-o-linear-gradient(#404040,#636363);
    background:-moz-linear-gradient(#404040,#636363);
    background:linear-gradient(#404040,#636363);
    box-shadow:3px 3px 8px #000000 inset;
    border-radius: 4px;
    padding-left:10px;
    padding-right:10px;
    padding-bottom:10px;
   }
```

- **Punto 6**

  Esta clase contiene el título del formulario.

  Clase de la aplicación: login-validation

  Clase de BootStrap: text-center

```
.login-validation {
    border-bottom: 4px solid #6c6c6c;
    width:90%;
   }

.login-validation h4 {
    color:#fff;
    border-radius: 8px;
    font-weight: bold;
   }
```

- **Punto 7**

  Contenedor de los elementos label e input del formulario.

  Clase de la aplicación: form-elements

  Clase de BootStrap: text-left, form-inline

  Clase híbrida: form-group

  Las clase form-group viene con estilos predeterminados de Bootstrap para los elementos label e input.

  Le hemos agregado estilos propios de la aplicación para las etiquetas label del formulario.

```
.form-row-login .form-group label {
    color:#fff;
   }
```

- **Punto 8**

  Esta clase contiene el botón del login.

  Clase de la aplicación: button-login

```
.button-login {
    background:linear-gradient(#ffffff,#d8d8d8);
    background:-o-linear-gradient(#ffffff,#f8f8f8);
    background:-moz-linear-gradient(#ffffff,#d8d8d8);
    background:linear-gradient(#ffffff,#d8d8d8);
    border:1px solid;
    border-color: #777777;
    border-radius: 8px;
    font-size:14px;
    color: #4e688d
   }

.button-login:hover {
   background:linear-gradient(#d8d8d8,#ffffff);
   background:-o-linear-gradient(#d8d8d8,#ffffff);
   background:-moz-linear-gradient(#d8d8d8,#ffffff);
   background:linear-gradient(#d8d8d8,#ffffff);
   border:1px solid;
   border-color: #999999;;
   border-radius: 8px;
   }
```

  El botón contiene un icono vectorial.

- **Punto 9**

  Aquí se mostrará el mensaje de aviso en el caso de introducir incorrectamente el login.

  Clase de la aplicación: status-message

  Clase de BootStrap: text-center, bg-danger

Clase híbrida: form-group

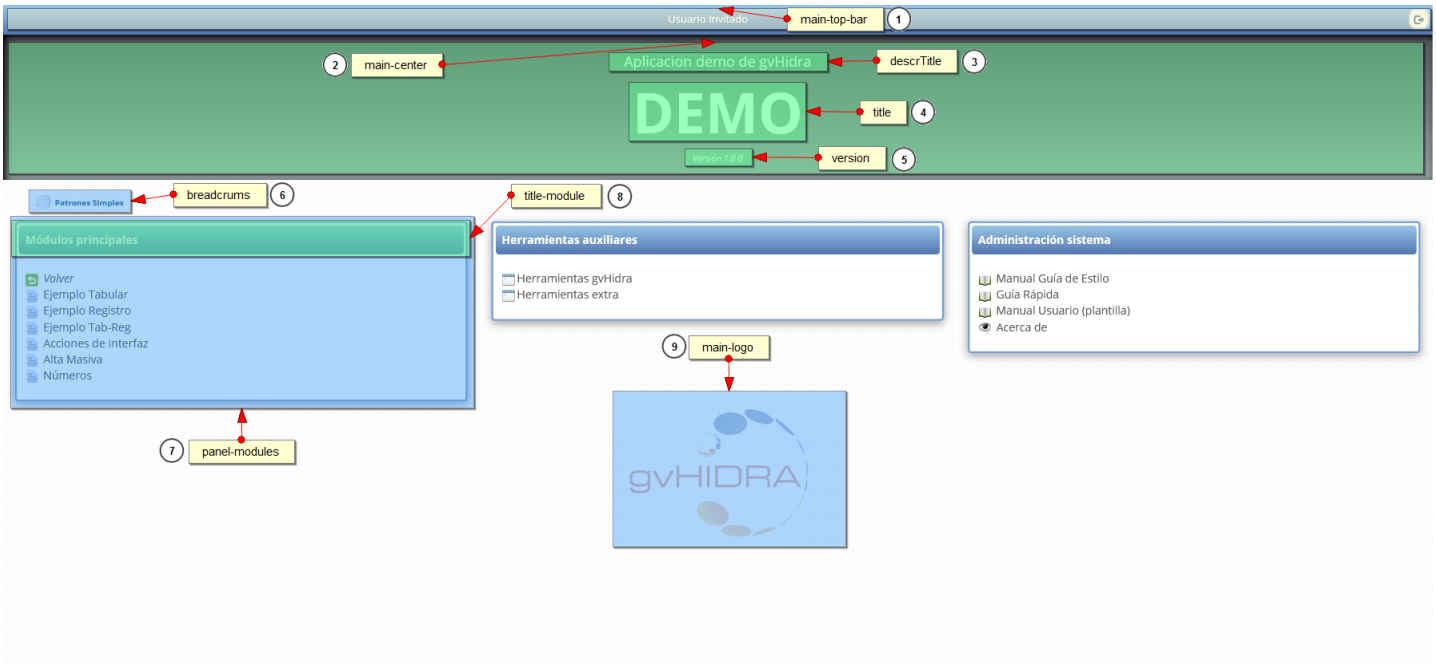
Las clase form-group viene con estilos predeterminados de Bootstrap para los elementos label e input.

Le hemos agregado estilos propios de la aplicación para las etiquetas label del formulario..

```
.status-message {
    color: #ED1B24;
    font-size: 18px;
   }
```

7.5.10. Pantalla de inicio

Los estilos de esta pantalla se cargan en la hoja de estilos aplicacion.css.



- **Punto 1**

  Contenedor de la barra superior del panel principal.

  Clase de la aplicación: main-top-bar

  Clase de BootStrap: text-center, pull-right

```
.main-top-bar {
    color:#fff;
    background:linear-gradient(#98c6e7,#5376b0);
    background:-o-linear-gradient(#98c6e7,#5376b0);
    background:-moz-linear-gradient(#98c6e7,#5376b0);
    background:linear-gradient(#98c6e7,#5376b0);
    padding: 8px 5px 5px;
   }
```

Para mantener el botón de salir a la derecha de la barra se a usado la clase de Bootstrap pull-right. Para colocar el campo usuario en el centro de la barra se ha usado la clase de Bootstrap text-center.

Para colocar el campo usuario en el centro de la barra se ha usado la clase de Bootstrap text-center.

- **Punto 2**

  Clase de la aplicación: main-center

  Clase de BootStrap: text-center

  La clase main-center nos da dos opciones, la primera es insertar un color o degradados para el fondo del contenedor, la segunda es insertar una imagen como fondo.

  Primera opción:

```
.main-center {
    background:linear-gradient(#404040,#929D9F);
    background:-o-linear-gradient(#404040,#929D9F);
    background:-moz-linear-gradient(#404040,#929D9F);
    background:linear-gradient(#404040,#929D9F);
    box-shadow:3px 3px 8px #000000 inset;
    margin-bottom: 14px;
    padding:20px;
   }
```

  Segunda opción:

```
.main-center{
    background: url("../images/logos/fondo-sup.jpg") 50% 50% no-repeat;
    background-size: cover;
    box-shadow:3px 3px 8px #000000 inset;
    margin-bottom: 14px;
    padding:20px;
    width: 100%;
   }
```

  Dentro de este contenedor existen tres contenedores diferentes, uno para la descripción, otro para el título y otro para la versión que corresponderán a los tres siguientes puntos.

- **Punto 3**

  Hace referencia al título largo de la aplicación.

  Identificador de la aplicación: descrTitle

  Cambiamos tamaño y color:

```
#descrTitle {
    font-size: 18px;
    color: #fff;
   }
```

- **Punto 4**

  Hace referencia al título corto de la aplicación.

Identificador de la aplicación: title

Cambiamos tamaño, color, tipo de letra:

```
#title  {
    color: #fff;
    font-size: 70px;
    font-weight: bold;
   }
```

- **Punto 5**

  Hace referencia a la versión de la aplicación.

  dentificador de la aplicación: version

  Cambiamos tamaño, color y estilo de letra de la versión

```
#version  {
    font-size: 11px;
    color: #fff;
    font-style: italic;
   }
```

- **Punto 6**

  Los estilos de este punto están definidos en breadcrumbs.css.

  Este contenedor contiene las migas de pan

  Identificador de la aplicación: breadcrumbs

- **Punto 7**

  Contenedor que contiene un modulo.

  Clase de la aplicación: panel-modules

  Cambiamos tamaño, color y estilo de letra de la versión

```
.panel-modules {
    border-style: solid;
    border-color: #759DCB;
    border-radius: 4px;
    box-shadow: 0px 3px 10px #999999;
   }
```

- **Punto 8**

  Contiene el título del modulo.

  Clase de la aplicación: title-module

  Cambiamos tamaño, color y estilo de letra de la versión

```
.main-modules .col-md-4 .title-module {
```

```
    background:linear-gradient(#98c6e7,#5376b0);
    background:-o-linear-gradient(#98c6e7,#5376b0);
    background:-moz-linear-gradient(#98c6e7,#5376b0);
    background:linear-gradient(#98c6e7,#5376b0);
    color:#fff;
    border-radius: 4px;
    font-size: 14px;
    padding:8px;
    font-weight: bold;
    margin: 2px;
   }
```

- **Punto 9**

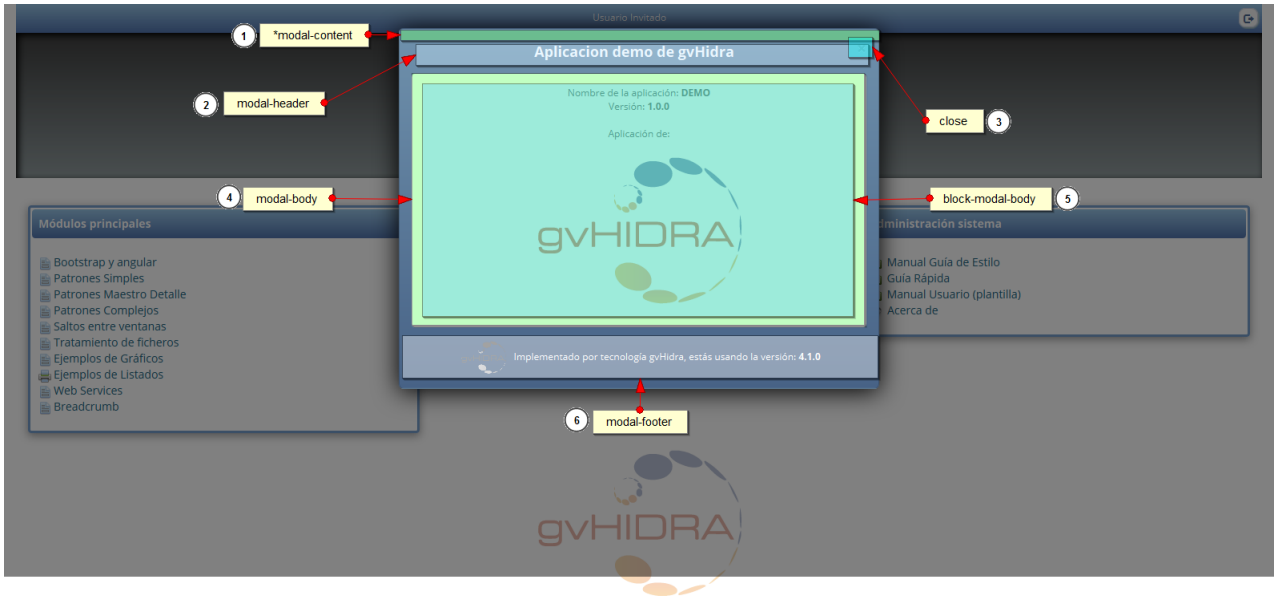
  Contenedor del logo de la aplicación donde se le ha aplica transparencia.

  Clase de la aplicación: main-logo

  Clase de BootStrap: text-center

```
.main-logo img {
    opacity: 0.4;
    filter: alpha(opacity=40);
   }
```

7.5.11. Acerca de



- **Punto 1**

  Contenedor que engloba toda la ventana.

  Clase de Bootstrap: text-center

  Clase híbrida: modal-content

```
.modal-content {
    background: linear-gradient(#6f91ac, #476286) repeat scroll 0 0 rgba(0, 0,
 0, 0);
    box-shadow: 0 3px 30px #000000;
   }
```

- **Punto 2**

  Contenedor que muestra el encabezado de la ventana.

  Clase híbrida: modal-header

  Estilo incluido en nuestra aplicación:

```
.modal-header {
    border-bottom: 0 none;
    color: #fff;
   }
```

- **Punto 3**

  Botón que cierra la ventana.

  Clase de Bootstrap: close

  Los estilos para dicho botón que ofrece Bootstrap son los siguientes:

```
.modal-header .close {
    margin-top: -2px;
   }

.button.close {
    background: none repeat scroll 0 0 transparent;
    border: 0 none;
    cursor: pointer;
    padding: 0;
   }

.close {
    color: #000;
    float: right;
    font-size: 21px;
    font-weight: bold;
    line-height: 1;
    opacity: 0.2;
    text-shadow: 0 1px 0 #fff;
   }
```

  En el caso de querer modificar estos estilos se habrá que definir en nuestro custom estas clases para reemplazar las predeterminadas de Bootstrap.

- **Punto 4**

  Contenedor del cuerpo de la ventana.

Clase híbrida: modal-body

Estilos añadidos desde la aplicación:

```
.modal-body {
    background-color: #fff;
    border-radius: 4px;
    margin: 0 auto;
    width: 95%;
   }
```

- **Punto 5**

  Color de la letra del cuerpo principal de la ventana.

  Clase de la aplicación: block-modal-body

  Estilos añadidos desde la aplicación:

```
.block-modal-body {
    color: #333333;
   }
```

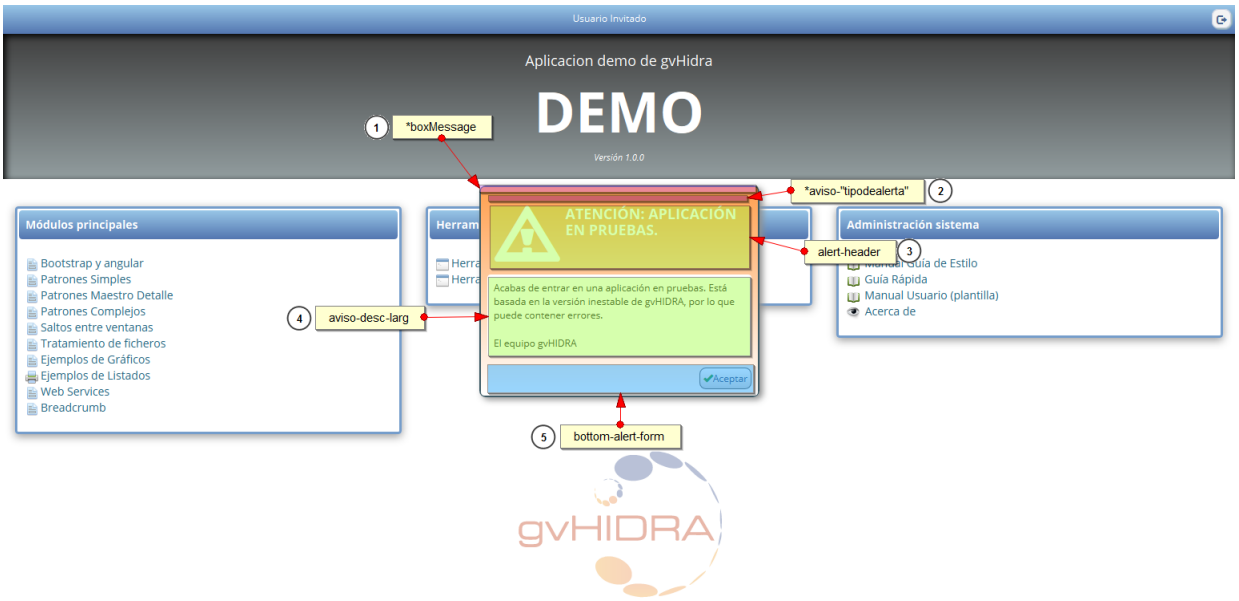
- **Punto 6**

  Contenedor que contiene el footer.

  Clase híbrida: modal-footer

```
.modal-footer {
    border-top:none;
    color:#fff;
   }

.modal-footer img {
    width: 64px;
   }
```

7.5.12. Ventanas de aviso



- **Punto 1**

  Contenedor de la ventana aviso.

  Clase de la aplicación: boxMessage

```
.boxMessage {
    background-color: #e5ecef;
    border: 2px solid #3c5964;
    border-radius: 8px;
    box-shadow: 4px 3px 7px 0 #8f9090;
    text-align: left;
    width: 350px;
   }
```

- **Punto 2**

  Clase para personalizar el color de las diferentes ventanas de aviso.

  Clase de la aplicación: aviso-"tipodealerta"

  Existen 5 tipos de ventanas emergentes para comunicar información, cada una de estas ventanas posee una clase específica y son las siguientes:

  aviso-alerta

  aviso-error

  aviso-sugerencia

  aviso-aviso

  aviso-confirm

Cada clase posee unos colores propios:

```
.aviso-alerta {
    background:linear-gradient(#ffa050,#ffffff);
    background:-o-linear-gradient(#ffa050,#ffffff);
    background:-moz-linear-gradient(#ffa050,#ffffff);
    background:linear-gradient(#ffa050,#ffffff);
   }

.aviso-error {
    background:linear-gradient(#D0240C,#ffffff);
    background:-o-linear-gradient(#D0240C,#ffffff);
    background:-moz-linear-gradient(#D0240C,#ffffff);
    background:linear-gradient(#D0240C,#ffffff);
   }

.aviso-sugerencia {
    background:linear-gradient(#5F9A2E,#ffffff);
    background:-o-linear-gradient(#5F9A2E,#ffffff);
    background:-moz-linear-gradient(#5F9A2E,#ffffff);
    background:linear-gradient(#5F9A2E,#ffffff);
   }

.aviso-aviso {
    background:linear-gradient(#3475D9,#ffffff);
    background:-o-linear-gradient(#3475D9,#ffffff);
    background:-moz-linear-gradient(#3475D9,#ffffff);
    background:linear-gradient(#3475D9,#ffffff);
   }

.aviso-confirm {
    background:linear-gradient(#5F9A2E,#ffffff);
    background:-o-linear-gradient(#5F9A2E,#ffffff);
    background:-moz-linear-gradient(#5F9A2E,#ffffff);
    background:linear-gradient(#5F9A2E,#ffffff);
   }
```

- **Punto 3**

  Contenedor que ocupa la cabecera de la ventana del aviso.

  Clase de la aplicación: alert-header

  A su vez este contenedor se divide en dos partes, una parte para el icono vectorial y otra para el título del mensaje.

  La parte del glyphicon usa la siguiente clase:

  Clase de la aplicación: alert-glyphicon

```
.alert-glyphicon {
    font-size: 70px;
   }
```

  Y la parte del título utiliza la etiqueta h4 para personalizarse:

```
.alert-glyphicon {
    font-size: 70px;
   }
```

- **Punto 4**

  Contenedor que ocupa la descripción del aviso.

  Clase de la aplicación: aviso-desc-larg

```
.aviso-desc-larg {
    background-color:#fff;
    border-radius: 4px;
    color:#333333;
    margin-top: 16px;
    padding: 8px;
   }
```

- **Punto 4**

  Este contenedor ocupa la parte inferior de la ventana de aviso.

  Clase de la aplicación: bottom-alert-form

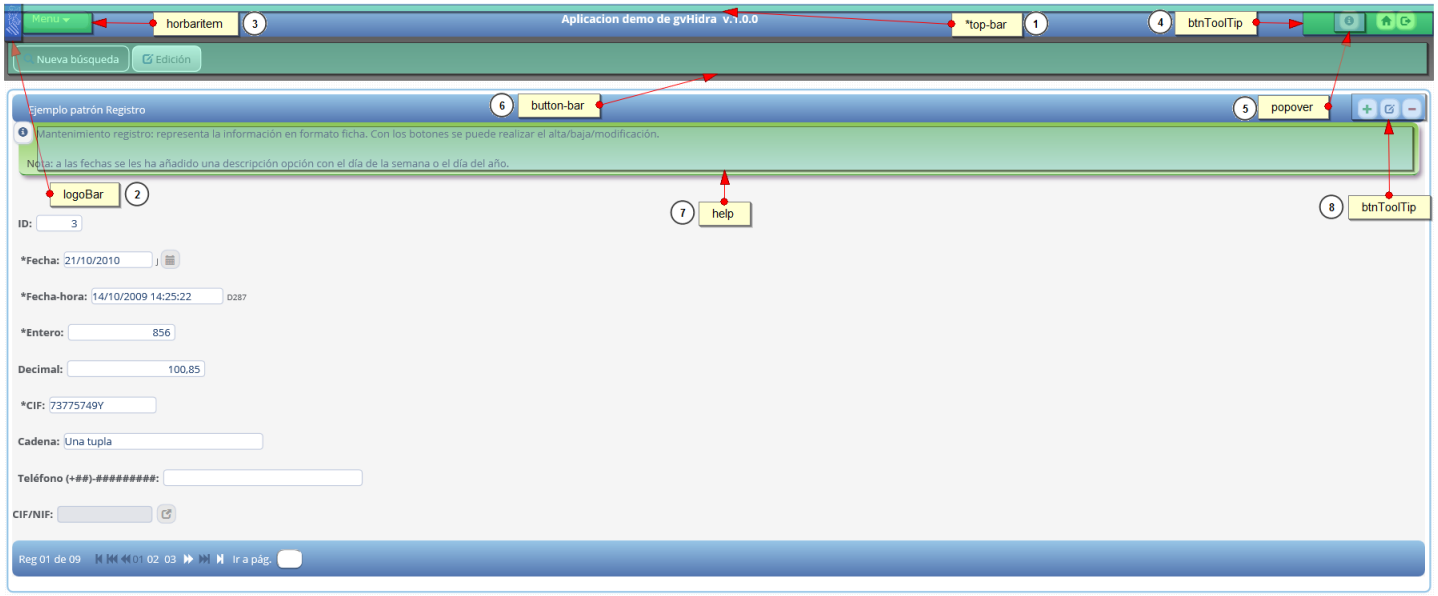
  Clase de BootStrap: text-right

  Los estilos para el botón aceptar son los siguientes:

```
.bottom-alert-form {
    background:linear-gradient(#ffffff,#d8d8d8);
    background:-o-linear-gradient(#ffffff,#f8f8f8);
    background:-moz-linear-gradient(#ffffff,#d8d8d8);
    background:linear-gradient(#ffffff,#d8d8d8);
    border:1px solid;
    border-color: #777777;
    border-radius: 8px;
    font-size:12px;
    padding:4px;
    color: #4e688d;
    margin-top: 14px;
   }

.bottom-alert-form {
    background:linear-gradient(#d8d8d8,#ffffff);
    background:-o-linear-gradient(#d8d8d8,#ffffff);
    background:-moz-linear-gradient(#d8d8d8,#ffffff);
    background:linear-gradient(#d8d8d8,#ffffff);
    border:1px solid;
    border-color: #999999;;
    border-radius: 8px;
   }
```

7.5.13. Partes genéricas de los paneles FIL LIS EDI

Todos los paneles tienen en común la parte superior de la pantalla y algunos elementos globales



- **Punto 1**

  Contenedor de la barra superior.

  Clase de la aplicación: top-bar

  Clase de Bootstrap: text-center

```
.top-bar {
    background: linear-gradient(#98c6e7,#5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    background: -moz-linear-gradient(#98c6e7, #5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    color: #fff;
    font-weight: bold;
    font-size: 14px;
    padding: 8px 8px 8px;
   }
```

- **Punto 2**

  Logo en miniatura de la aplicación.

  Clase/identificador de la aplicación: logoBar

  Estilos para la posición del logo:

```
.div#logoBar {
       margin-left: -21px;
```

```
       margin-top: -6px;
       padding-right: 4px;
       position: absolute;
   }
```

Estilos para el tamaño del logo:

```
.logoBar {
       border-style: none;
       height: 37px;
       width: 19px;
   }
```

- **Punto 3**

Los estilos de este punto están definidos en Layersmenu-cit.css

Botón que ofrece un menú en la barra superior.

Clase de la aplicación: horbaritem

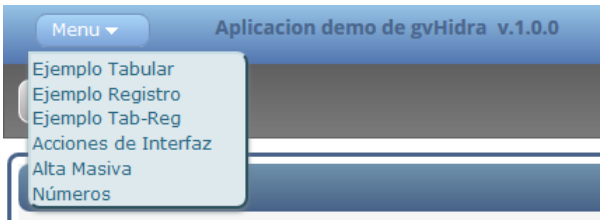
Estilos para el botón:

```
.horbaritem {
    background:linear-gradient(#a9d5f7,#7ea4d9);
    background:-o-linear-gradient(#a9d5f7,#7ea4d9);
    background:-moz-linear-gradient(#a9d5f7,#7ea4d9);
    background:linear-gradient(#a9d5f7,#7ea4d9);
    border:1px solid;
    border-color:#7593ba;
    border-radius:8px;
    color: #FFFFFF;
    float: left;
    font-family: verdana, arial, helvetica, sans-serif;
    font-size: 12px;
    padding: 2px 10px;
    white-space: nowrap;
   }
```

Estilos del botón hover:

```
.horbaritem:hover {
    background:linear-gradient(#7ea4d9,#a9d5f7);
    background:-o-linear-gradient(#7ea4d9,#a9d5f7);
    background:-moz-linear-gradient(#7ea4d9,#a9d5f7);
    background:linear-gradient(#7ea4d9,#a9d5f7);
   }
```

Cuando mantienes el cursor sobre el menú aparece la siguiente ventana:



En la anterior imagen vemos que aparece una ventana y en ella enlaces.

Para personalizar la ventana usamos la clase de la aplicación subframe.

```
.subframe {
    background-color: #dee9ed;
    border: 2px outset #5093ac;
    border-radius: 0 8px 8px;
    box-shadow: 0 2px 3px 0 #758088;
    display: block;
    padding-top: 2px;
    position: relative;
   }
```

Para personalizar los enlaces usamos la clase de la aplicación item.

```
.item {
    position: relative;
    text-align: left;
    white-space: nowrap;
    color: #296883;
    font-family: verdana, arial, helvetica, sans-serif;
    font-size: 12px;
   }

.item a:link  {
    color: #296883;
   }

.item a:visited {
    color: #296883;
   }

.item a:hover  {
    color: #0d4421;
    background-color: #b8cbbd;
   }

.item a:active   {
    color: #ff0000;
   }
```

- **Punto 4**

Botones de la barra superior.

Clase de la aplicación: btnToolTip

Diseño de los botones con iconos vectoriales

```
.btnToolTip  {
    padding: 5px;
    font-size: 12px;
    font-weight: normal;
    cursor:pointer;
    border: thin #C1CADD solid;
    border-radius: 8px 8px 8px 8px;
    border-radius: 8px 8px 8px 8px;
    box-shadow: #b8cbbd 1px 1px 1px 0px;
    box-shadow: #b8cbbd 1px 1px 1px 0px;
    background-color: #f4f4f4;
   }

.btnToolTip:hover  {
    background: linear-gradient(#d8d8d8, #ffffff);
    background: -o-linear-gradient(#d8d8d8, #ffffff);
    background: -moz-linear-gradient(#d8d8d8, #ffffff);
    background: linear-gradient(#d8d8d8, #ffffff);
   }
```

- **Punto 5**

Cuando se pulsa el botón de información de la barra superior saldrá la siguiente ventana.

Esta ventana usa clases de Bootstrap.

Clases para el título de la ventana:

Clase de la aplicación: panel-info

Clase híbrida: popover-title

```
.popover-title {
    background:linear-gradient(#6f91ac,#476286);
    background:-o-linear-gradient(#6f91ac,#476286);
    background:-moz-linear-gradient(#6f91ac,#476286);
    background:linear-gradient(#6f91ac,#476286);
    color:#fff;
   }
```

El cuerpo de la ventana que usa la clase de Bootstrap “popover-content” contiene una tabla que muestra la información de los campos, esta tabla contiene la clase de Bootstrap table-striped para colorear las filas impares de la tabla.

- **Punto 6**

Contenedor que contiene la barra de botones de control de los paneles.

Clase de la aplicación: button-bar

```
.button-bar {
    margin-bottom:10px;
    padding: 10px 10px 10px;
    background: linear-gradient(#606060, #7b7b7b);
    background: -o-linear-gradient(#606060, #7b7b7b);
    background: -moz-linear-gradient(#606060, #7b7b7b);
    background: linear-gradient(#606060, #7b7b7b);
    border: thin solid #606060;
   }
```

Los botones que se encuentran en dicho contenedor poseen dos estados diferentes que es activo e inactivo.

Cuando el botón se encuentra inactivo:

```
.tab.disabled,.tab[disabled] {
    display: inline-block;
    padding: 6px 12px;
    margin-bottom: 0;
    background: linear-gradient(#e4e4e4, #aaaaaa);
    background: -o-linear-gradient(#e4e4e4, #aaaaaa);
    background: -moz-linear-gradient(#e4e4e4, #aaaaaa);
    background: linear-gradient(#e4e4e4, #aaaaaa);
    border-radius: 8px;
    color: #55585D;
    cursor: not-allowed;
   }
```

Cuando el botón se encuentra activo:

```
.tab {
      display: inline-block;
      padding: 6px 12px;
      margin-bottom: 0;
    background: linear-gradient(#5c5c5c, #7f7f7f);
    background: -o-linear-gradient(#5c5c5c, #7f7f7f);
    background: -moz-linear-gradient(#5c5c5c, #7f7f7f);
    background: linear-gradient(#5c5c5c, #7f7f7f);
    border: 1px solid;
    border-color: #c6c6c6;
    border-radius: 8px;
    border: 1px solid #fff;
    color: #fff;
    cursor: pointer;
```

```
   }
```

- **Punto 7**

  Contenedor que contiene un mensaje de ayuda.

  Clase de la aplicación: help

```
.help {
    color: #444;
    margin: 0px 8px 10px 8px;
    padding: 5px 5px 5px 10px;
    border-radius: 8px 8px 8px 8px;
    border-radius: 8px 8px 8px 8px;
    box-shadow: #9f8da2 2px 2px 2px 0px;
    box-shadow: #9f8da2 4px 4px 4px 0px;
    background: linear-gradient(#8cc658, #ccedbc);
    background: -o-linear-gradient(#8cc658, #ccedbc);
    background: -moz-linear-gradient(#8cc658, #ccedbc);
    background: linear-gradient(#8cc658, #ccedbc);
    border-bottom: 2px solid #006c00;
   }
```

- **Punto 8**

  Botones que realizan acciones como insertar, borrar y editar.

  Clase de la aplicación: btnToolTip

```
.btnToolTip  {
    padding: 5px;
    font-size: 12px;
    font-weight: normal;
    cursor:pointer;
    border: thin #C1CADD solid;
    border-radius: 8px 8px 8px 8px;
    border-radius: 8px 8px 8px 8px;
    box-shadow: #b8cbbd 1px 1px 1px 0px;
    box-shadow: #b8cbbd 1px 1px 1px 0px;
    background-color: #f4f4f4;
   }
```

Estilos para el botón al pasar el ratón por encima.

```
.btnToolTip:hover  {
    background: linear-gradient(#d8d8d8, #ffffff);
    background: -o-linear-gradient(#d8d8d8, #ffffff);
    background: -moz-linear-gradient(#d8d8d8, #ffffff);
    background: linear-gradient(#d8d8d8, #ffffff);
   }
```

Cuando un botón es pulsado el resto de botones cambian su estado a “desactivado”.

```
.btnToolTip.disabled,.btnToolTip[disabled], fieldset[disabled] .btnToolTip  {
    pointer-events: none;
```

```
    cursor: not-allowed;
    filter: alpha(opacity=75);
    box-shadow: none;
    box-shadow: none;
    opacity: .75;
    background-color: #e2e2e2;
    color: #808080;
   }

.btnToolTip.off {
    visibility: hidden
   }
```

7.5.14. Partes genéricas de los paneles LIS EDI

- **Punto 1**

  Contenedor que contiene la barra superior del panel.

  Clase de la aplicación: topBar-panelPrimary

```
.topBar-panelPrimary {
    color: #fff;
    padding: 5px;
    text-align: right;
    background: linear-gradient(#98c6e7,#5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    background: -moz-linear-gradient(#98c6e7, #5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    border-radius: 8px 8px 8px 8px;
   }
```

- **Punto 2**

  Contiene la barra inferior del panel.

  Clase de la aplicación: bottomBar-panelPrimary

```
.bottomBar-panelPrimary   {
    color: #fff;
    padding: 5px;
    text-align: right;
    background: linear-gradient(#98c6e7,#5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    background: -moz-linear-gradient(#98c6e7, #5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    border-radius: 8px 8px 8px 8px;
   }
```

- **Punto 3**

  Contenedor que contiene el paginador.

  Clase de la aplicación: pagesPrimary

Clase de Bootstrap: col-xs-12, col-sm-8, col-md-6, text-left

En el caso de que el paginador utilize Glyphicon se podrá personalizar el color de estos cuando las flechas estén inactivas con el estilo color.

```
.pagesPrimary {
    color: #4a668e;
    padding:3px;
    text-align: left;
    border-radius: 8px 0px 8px 8px;
    border-radius: 8px 0px 8px 8px;
   }
```

Cuando las fechas del paginador esten activas se escogerá el color que se desee con el estilo color

```
.pagesPrimary a {
    text-decoration: none;
    color: #ffffff;
   }
```

Estilo hover de las fechas activas.

```
.pagesPrimary a:hover  {
    color: #223955;
   }
```

Para poner el texto informativo que rodea al paginador como el número de registro o el campo ir a página del color que queremos, usaremos la siguiente clase de la aplicación.

```
.paginador-pag  {
     color: #fff;
   }
```

- **Punto 4**

  Botón Guardar/Cancelar.

  Clase de la aplicación: btn_primary

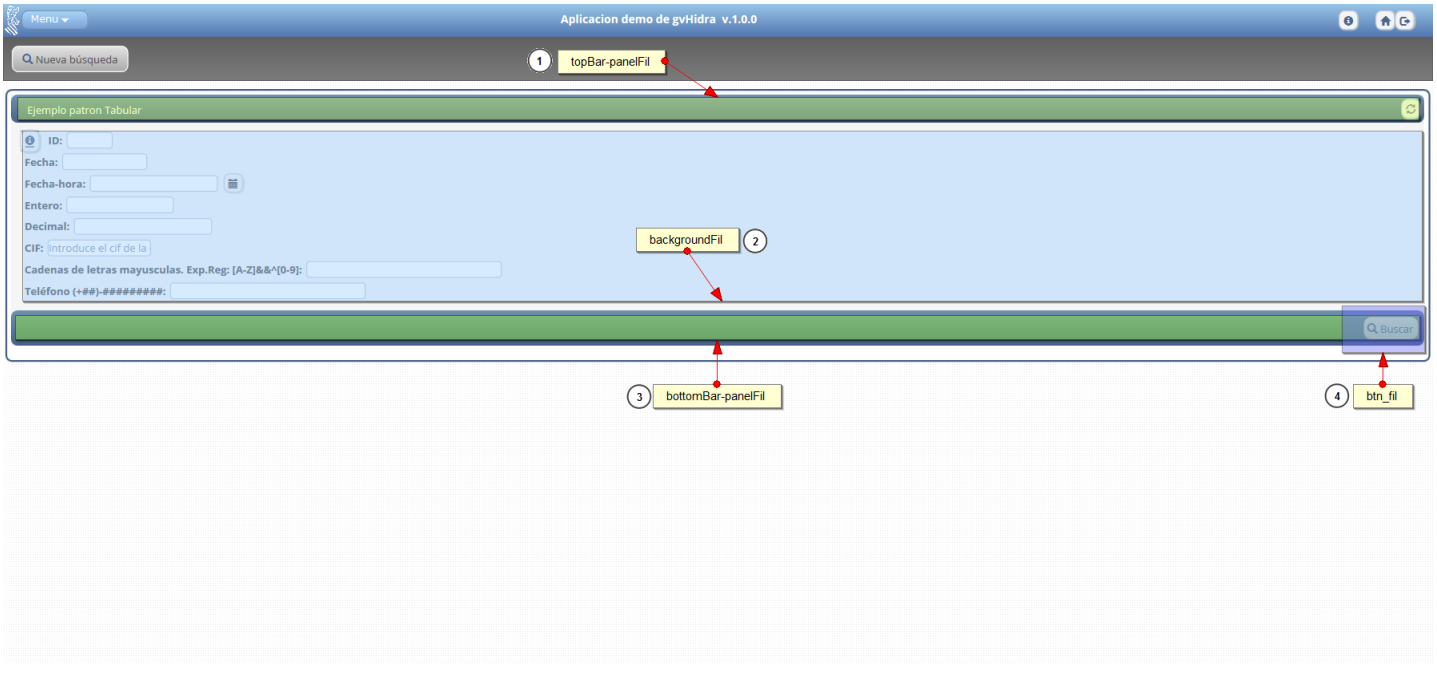
```
.btn_primary {
    color: #4a668e;
    background: linear-gradient(#ffffff, #d8d8d8);
    background: -o-linear-gradient(#ffffff, #d8d8d8);
    background: -moz-linear-gradient(#ffffff, #d8d8d8);
    background: linear-gradient(#ffffff, #d8d8d8);
    border: 1px solid;
    border-color: #777777;
    border-radius: 10px;
    padding: 5px 6px 4px 4px;
    margin: 2px;
   }

.btn_primary:hover {
    background: linear-gradient(#d8d8d8, #ffffff);
    background: -o-linear-gradient(#d8d8d8, #ffffff);
```

```
    background: -moz-linear-gradient(#d8d8d8, #ffffff);
    background: linear-gradient(#d8d8d8, #ffffff);
    border-radius: 10px;
    border: 1px solid;
    border-color: #999999;
   }
```

En el caso de que el botón tenga un glyphicon usará los siguiente estilos.

```
.glyphicon-ok:before {
    color: #42a650;
   }

.glyphicon-remove:before {
    color: #a74a43;
   }
```

7.5.15. Panel FIL



- **Punto 1**

  Contenedor que contiene la barra superior del panel de búsqueda (FIL).

  Clase de la aplicación: topBar-panelFil

```
.topBar-panelFil {
    color: #fff;
    text-align: right;
    padding: 5px;
    background: linear-gradient(#6f91ac,#476286);
    background: linear-gradient(#6f91ac,#476286);
    background: -moz-linear-gradient(#6f91ac, #476286);
    background: linear-gradient(#6f91ac,#476286);
```

```
    border-radius: 8px 8px 8px 8px;
   }

.titlePanel {
    display:inline;
    margin-top:4px;
   }
```

- **Punto 2**

  Contenedor que contiene el bloque central del panel.

  Clase de la aplicación: backgroundFil

  Estilos para los input del panel.

```
.backgroundFil input{
    border-radius:4px;
    margin: 2px 2px 4px;
   }
```

  Estilos para los campos del panel para que al seleccionarlos se muestre un borde de un color.

```
.backgroundFil input:focus {
    border: 2px solid #008FFF;
   }
```

- **Punto 3**

  Contiene la barra inferior del panel.

  Clase de la aplicación: bottomBar-panelFil

```
.bottomBar-panelFil {
    color: #fff;
    text-align: right;
    padding: 5px;
    background: linear-gradient(#6f91ac,#476286);
    background: linear-gradient(#6f91ac,#476286);
    background: -moz-linear-gradient(#6f91ac, #476286);
    background: linear-gradient(#6f91ac,#476286);
    border-radius: 8px 8px 8px 8px;
   }
```

- **Punto 4**

  Botón buscar.

  Clase de la aplicación: btn_fil

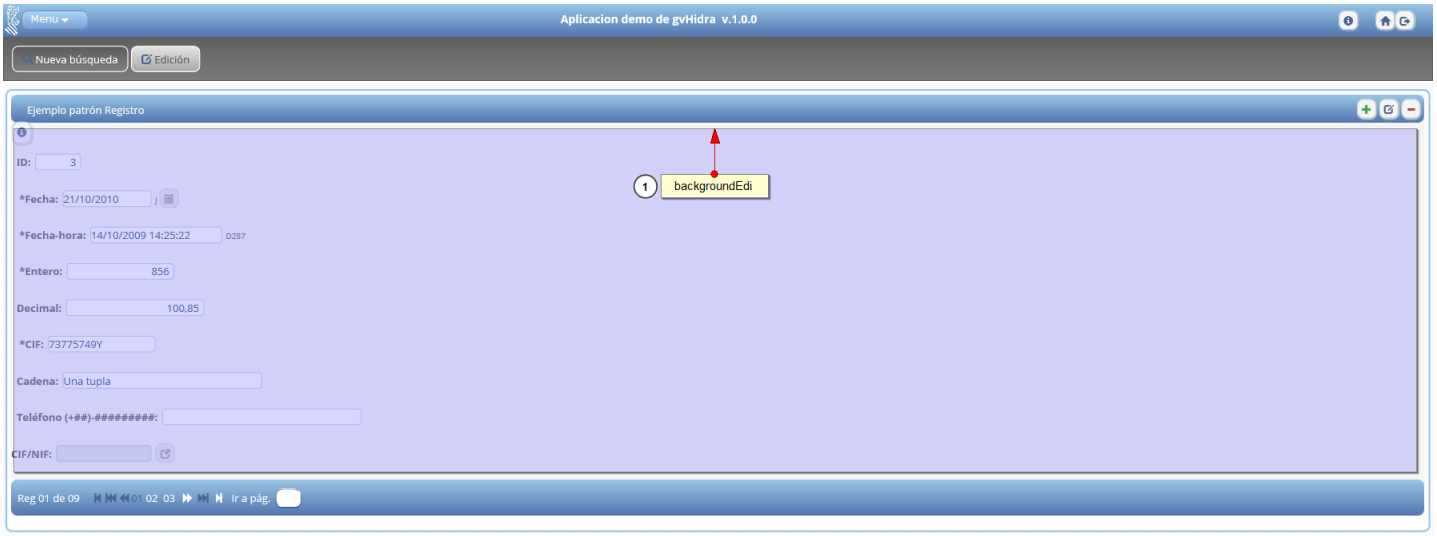
```
.btn_fil {
    color: #4a668e;
    background: linear-gradient(#ffffff, #d8d8d8);
    background: -o-linear-gradient(#ffffff, #d8d8d8);
    background: -moz-linear-gradient(#ffffff, #d8d8d8);
    background: linear-gradient(#ffffff, #d8d8d8);
```

```
    border: 1px solid;
    border-color: #777777;
    border-radius: 10px;
    padding: 5px 6px 4px 4px;
    margin: 2px;
   }

.btn_fil:hover {
    background: linear-gradient(#d8d8d8, #ffffff);
    background: -o-linear-gradient(#d8d8d8, #ffffff);
    background: -moz-linear-gradient(#d8d8d8, #ffffff);
    background: linear-gradient(#d8d8d8, #ffffff);
    border-radius: 10px;
    border: 1px solid;
    border-color: #999999;
   }
```

En el caso de que el botón tenga un glyphicon usará los siguiente estilos.

```
.glyphicon-search:before  {
    color: #4a668e;
    font-style: 24px;
   }
```

7.5.16. Panel EDI



- **Punto 1**

  Contenedor que contiene el bloque central del panel.

  Clase de la aplicación: backgroundEdi

```
.backgroundEdi input  {
```

```
    border-radius:4px;
    margin: 2px 2px 4px;
   }
```

Estilos para los campos del panel para que al seleccionarlos se muestre un borde de un color.

```
.backgroundEdi input:focus  {
     border: 2px solid #008FFF;
   }
```

- **Punto 2**

Clases que se utilizan dentro del panel registro/búsqueda.

Clase de la aplicación: edit

Se usa para los campos que son editables.

```
.edit  {
    color: #234372;
    border: 1px solid #C1CADD;
    background-color: #FFFFFF;
   }
```

Clase de la aplicación: noEdit

Se usa para los campos que no son editables.

```
.noEdit  {
    color: #234372;
    border: 1px solid #C1CADD;
    background-color: #E2E4E9;
   }
```

Clase de la aplicación: new

Se aplica cuando el registro pasa a estado insertar.

```
.new {
    color: #64799B;
    border: 1px solid #C1CADD;
    background-color: #E2E4E9;
   }
```

Clase de la aplicación: modify

Se aplica cuando el registro pase a estado editable

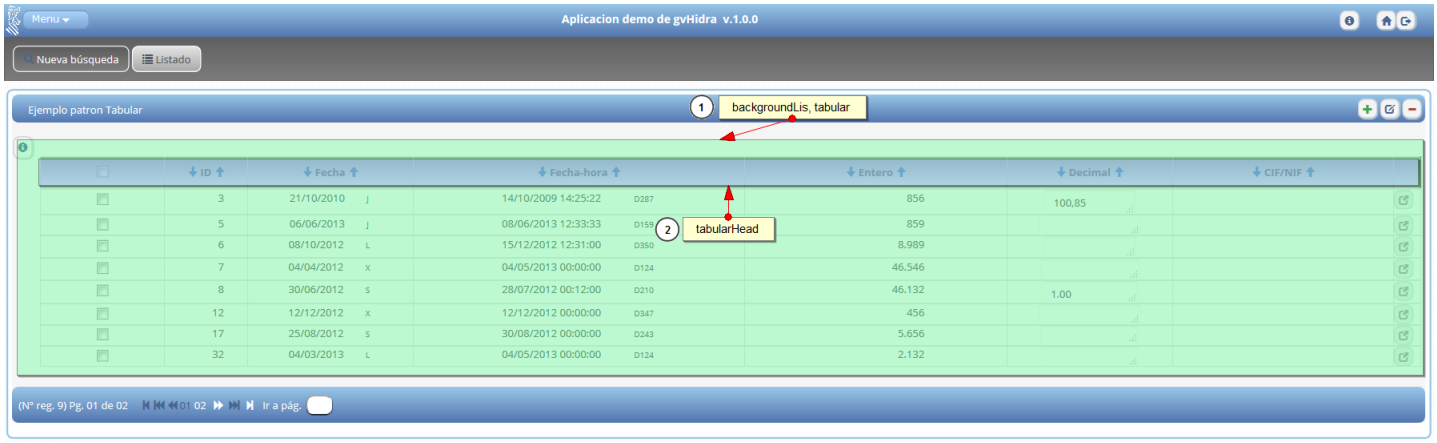
```
.modify  {
    background-color: #ffffff;
    font-weight: bold;
    color: #234372;
    border: 1px solid #c1cadd;
    background-color: #FFFFFF;
   }
```

Clase de la aplicación: delete

Se aplica cuando el registro pase a estado eliminado

```
.delete {
    color: #CBCBCB;
   }
```

7.5.17. Panel LIS



- **Punto 1**

  Contenedor que contiene la tabla de un panel tabular.

  Clase de la aplicación: backgroundLis, tabular

  Clase de Bootstrap: table-responsive, table, table-bordered

- **Punto 2**

  Contiene el encabezado de la tabla.

  Clase de la aplicación: tabularHead

```
.tabularHead  {
    background: linear-gradient(#aaaaaa, #ffffff);
    background: -o-linear-gradient(#aaaaaa, #ffffff);
    background: -moz-linear-gradient(#aaaaaa, #ffffff);
    background: linear-gradient(#aaaaaa, #ffffff);
    border: 1px solid;
    padding: 4px;
             border-bottom: 2px solid #999999;
   }
```

Color de los campos del encabezado.

```
.tabularHead label  {
     color: #626262;
   }
```

- **Punto 3**

Clases que se utilizan dentro del panel tabular.

Clase de la aplicación: oodRecord

Estilo aplicable a las filas impares de la tabla.

```
.oddRecord {
    background-color: #F5F5F5;
   }
```

Clase de la aplicación: evenRecord

Estilo aplicable a las filas pares de la tabla.

```
.evenRecord {
    background-color: #F5F5F5;
   }
```

Clase de la aplicación: rowOn

Estilo aplicable para cuando el ratón está encima una fila.

```
.rowOn {
    background-color: #DFDFDF;
   }
```

Clase de la aplicación: rowDeleted

Estilo aplicable para cuando la fila está marcada para ser borrada.

```
.rowDeleted {
    background-color: #F5F5F5;
   }
```

Clase de la aplicación: tableDelete

Estilos que se aplican al borrar una fila, como por ejemplo el color del texto.

```
.tableDelete {
    color: #CBCBCB;
   }
```

Clase de la aplicación: tableInsert

Estilos que se aplican al insertar una fila, como por ejemplo el color del texto, fondo.

```
.tableInsert {
    font-weight: bold;
```

```
    color: #234372 !important;
    border: 1px solid #C1CADD !important;
    background-color: #FFFFFF !important;
   }
```

Clase de la aplicación: tableNew

```
.tableNew {
    background-color: #F5F5F5;
   }
```

Clase de la aplicación: tableModify

Estilos que se aplican al editar una fila, como por ejemplo el color del texto o el borde.

```
.tableModify {
    font-weight: bold;
    color: #234372;
    border: 1px solid #C1CADD;
   }
```

Clase de la aplicación: tableEdit

Se usa para los campos de una tabla que son editables.

```
.tableEdit {
    background-color: #F5F5F5;
   }
```

Clase de la aplicación: tableNoEdit

Se usa para los campos de una tabla que no son editables.

```
.tableNoEdit {
    background-color: #F5F5F5;
   }
```

Como se ha explicado en el capítulo 3 punto Mensajes y Errores [107], se clasifican los mensajes en cuatro tipos.

	Código	Descripción
☐	01	DICCIONARIO
☐	02	MANUAL
☐	03	TUDIOS2
☐	04	LEGISLACION
☐	05	JURISPRUDENCIA
☐	06	TRATADO

Clase de la aplicación: alertRow

Este estilo corresponde al de "alerta".

```
.alertRow {
    background-color: #FFE5C4;
   }
```

Clase de la aplicación: noticeRow

Este estilo corresponde al de "aviso".

```
.noticeRow {
    background-color: #A9C7DA;
   }
```

Clase de la aplicación: errorRow

Este estilo corresponde al de "error".

```
.errorRow {
    background-color: #F1C2C5;
   }
```

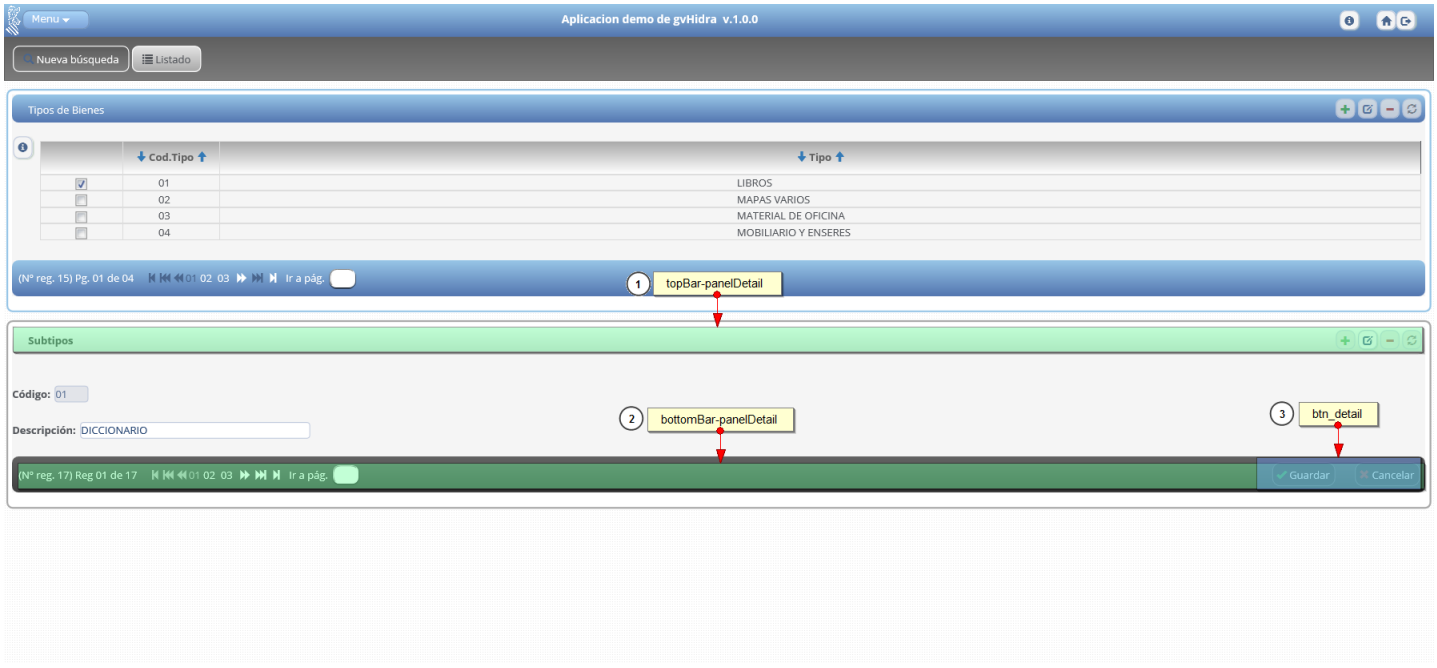
Clase de la aplicación: suggestionRow

Este estilo corresponde al de "sugerencia".

```
.suggestionRow {
    background-color: #ced6b2;
   }
```

7.5.18. Panel maestro-detalle

Este apartado engloba al panel detalle, ya sea un panel EDI o un panel LIS



- **Punto 1**

  Contiene la barra superior del panel.

  Clase de la aplicación: topBar-panelDetail

```
.topBar-panelDetail {
```

```
    color: #444;
    padding: 5px;
    font-weight: bold;
    text-align: right;
    border-bottom: thin solid #aaa;
    background: linear-gradient(#ffffff, #d8d8d8);
    background: -o-linear-gradient(#ffffff, #f8f8f8);
    background: -moz-linear-gradient(#ffffff, #d8d8d8);
    background: linear-gradient(#ffffff, #d8d8d8);
    border-radius: 8px 8px 8px 8px;
   }
```

- **Punto 2**

  Contiene la barra inferior del panel.

  Clase de la aplicación: bottomBar-panelDetail

```
.bottomBar-panelDetail {
    color: #fff;
    padding: 5px;
    text-align: right;
    background: linear-gradient(#636363, #404040);
    background: -o-linear-gradient(#636363, #404040);
    background: -moz-linear-gradient(#636363, #404040);
    background: linear-gradient(#636363, #404040);
    border-radius: 8px 8px 8px 8px;
   }
```

- **Punto 3**

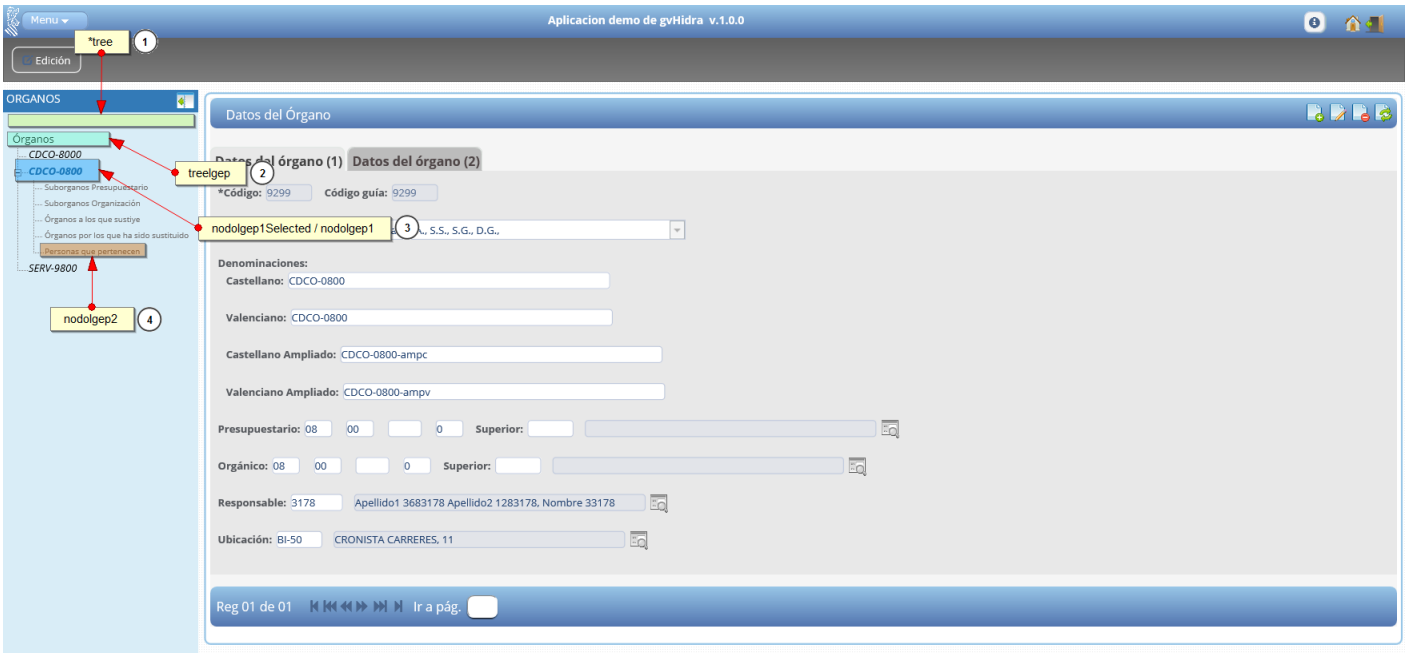
  Da estilos a los botones de la barra inferior.

  Clase de la aplicación: btn_detail

```
.btn_detail {
    background: #444;
    border: 1px solid;
    border-color: #777777;
    border-radius: 10px;
    padding: 5px 6px 4px 4px;
    margin: 2px;
   }

.btn_detail:hover  {
    background: linear-gradient(#7f7f7f, #5c5c5c);
    background: -o-linear-gradient(#7f7f7f, #5c5c5c);
    background: -moz-linear-gradient(#7f7f7f, #5c5c5c);
    background: linear-gradient(#7f7f7f, #5c5c5c);
   }
```

7.5.19. Panel árbol

Los estilos de este panel se encontran en tree.css



- **Punto 1**

  Apariencia del panel inicial de un árbol (que está en la parte izquierda de la ventana), en este punto podremos cambiar su color de fondo.

  Clase de la aplicación: tree

```
.tree {
    background-color: #D9EDF7;
  }
```

- **Punto 2**

  Controlar color, tamaño, background del nodo raíz del árbol.

  Clase de la aplicación: treeIgep

- **Punto 2**

  Controlar apariencia del nodo raíz dependiendo del su estado (visitado: visited , no visitado: link y cuando se pasa el ratón por encima sin pinchar : hover).

  Clase de la aplicación: .treeIgep a , .treeIgep a:link , .treeIgep a:visited , .treeIgep a:hover

- **Punto 2**

  Lo mismo que en el caso anterior, pero solo se aplica cuando el nodo raíz ha sido ya seleccionado y expandido. Por ejemplo, cuando se pasa el ratón por encima del nodo raíz, una vez este ha sido seleccionado y expandido, se aplica entonces el código que está en la sección .treeIgepSelected a:hover , y no el de treeIgep a:hover.

  Clase de la aplicación: .treeIgepSelected a , .treeIgepSelected a:link , .treeIgepSelected a:visited , .treeIgepSelected a:hover

- **Punto 3**

Igual para el caso anterior (que se aplicaba solo al nodo raíz), pero se aplica en este caso solo a los nodo del árbol de primer nivel (solo los que aparecen al expandir el nodo raíz).

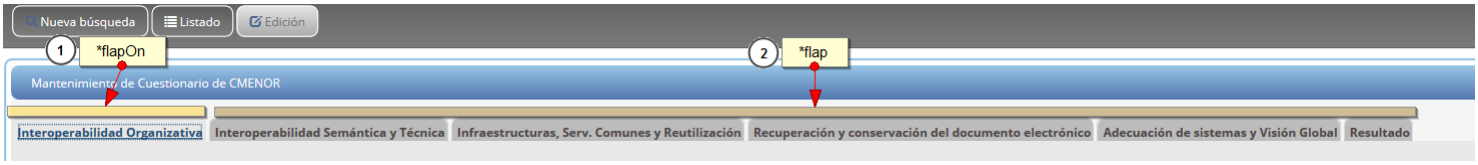
Clase de la aplicación: .nodoIgep1 y nodoIgep1Selected (con sus posibles opciones a:link a:hover ...etc)

- **Punto 4**

  Igual que nodoIgep1, pero aplicado en este caso solo a los nodos del segundo nivel y posteriores del árbol.

  Clase de la aplicación: .nodoIgep2 y nodoIgep2Selected (con sus posibles opciones a:link a:hover ...etc)

7.5.20. Solapas



Las solapas tienes dos estados, activas e inactivas.

- **Punto 1**

  Clase de la aplicación: flapOn

  Solapa activa:

```
.flapOn {
    top:0px;
    left:10px;
    margin-right: 3px;
    margin-bottom: 0px;
    padding-left: 5px;
    padding-right: 2px;
    padding-top: 5px;
    border-left:2px solid #e1e8e3;
    border-right:2px solid #e1e8e3;
    border-top:2px solid #e1e8e3;
    border-radius: 8px 8px 0px 0px;
    border-radius: 8px 8px 0px 0px;
    background: #e8e8e8;
    float:left;
    z-index:0;
    cursor: not-allowed;
   }

optionFlapOn {
    font-weight: bold;
    color: #444;
    text-decoration: none;
    cursor: not-allowed;
```

```
   }

.optionFlapOn:hover {
    text-decoration: none;
    cursor: not-allowed;
    color: #444;
   }
```

- **Punto 2**

  Clase de la aplicación: flap

  Solapa inactiva:

```
.flap {
    top:0px;
    left:10px;
    margin-right: 3px;
    margin-bottom: 0px;
    padding-left: 5px;
    padding-right: 2px;
    padding-top: 5px;
    border-left:2px solid #e1e8e3;
    border-right:2px solid #e1e8e3;
    border-top:2px solid #e1e8e3;
    background-color: #bcbcbc;
    border-radius: 8px 8px 0px 0px;
    border-radius: 8px 8px 0px 0px;
    cursor: pointer;
    float:left;
    z-index:0;
   }

.flap:hover {
    background: linear-gradient(#5C5C5C,#cacaca);
   }

.optionFlap {
    color: #4F4647;
    font-weight: bold;
    text-decoration: none;
   }

.optionFlap:hover {
    text-decoration: none;
    color: #fff;
   }
```

7.5.21. Debugger

- **Punto 1 y 2**

  Clase para los efectos de las dos barras que contienen títulos.

  Identificador de la aplicación: titleDebugger

```
#titleDebugger {
    background:linear-gradient(#404040,#929D9F);
    background:-o-linear-gradient(#404040,#929D9F);
    background:-moz-linear-gradient(#404040,#929D9F);
    background:linear-gradient(#404040,#929D9F);
    color:#fff;
    font-size:22px;
    padding:4px;
   }
```

- **Punto 3**

  Contenedor de la primera barra de administración.

  Identificador de la aplicación: panelDebugger

```
#panelDebugger {
    background:linear-gradient(#98c6e7,#5376b0);
    background:-o-linear-gradient(#98c6e7,#5376b0);
    background:-moz-linear-gradient(#98c6e7,#5376b0);
    background:linear-gradient(#98c6e7,#5376b0);
    padding: 4px;
    color:#fff;
   }
```

- **Punto 4**

  Contenedor de la segunda barra de administración.

  Clase de la aplicación: paramsDebugger

```
.paramsDebugger {
    background: linear-gradient(#8cc658, #ccedbc);
    background: -o-linear-gradient(#8cc658, #ccedbc);
```

```
    background: -moz-linear-gradient(#8cc658, #ccedbc);
    background: linear-gradient(#8cc658, #ccedbc);
   }
```

Clase para los efectos de los botones del panel debugger.

```
.buttonDebugger {
    padding-left:10px;
    padding-right:10px;
    padding-top:2px;
    padding-bottom:2px;
    color: #fff;
    background:linear-gradient(#a9d5f7,#7ea4d9);
    background:-o-linear-gradient(#a9d5f7,#7ea4d9);
    background:-moz-linear-gradient(#a9d5f7,#7ea4d9);
    background:linear-gradient(#a9d5f7,#7ea4d9);
    border:1px solid;
    border-color:#7593ba;
    border-radius:8px;
   }

.buttonDebugger:hover {
    background:linear-gradient(#7ea4d9,#a9d5f7);
    background:-o-linear-gradient(#7ea4d9,#a9d5f7);
    background:-moz-linear-gradient(#7ea4d9,#a9d5f7);
    background:linear-gradient(#7ea4d9,#a9d5f7);
   }
```

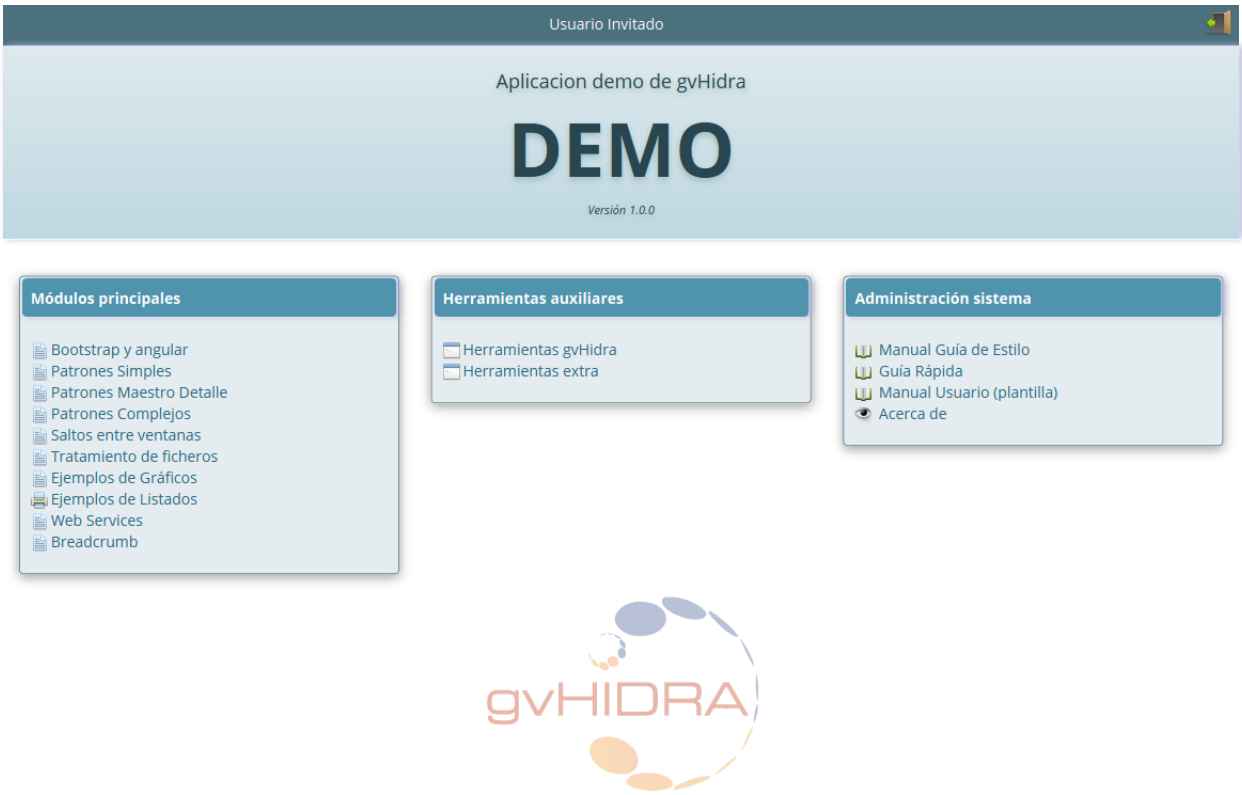
7.5.22. Ejemplos prácticos

Modificaremos la pantalla entrada con los estilos actuales para que se parezca lo más posible a los estilos clásicos de las versiones anteriores de gvHidra.

7.5.22.1. Pantalla principal

Pantalla actual:

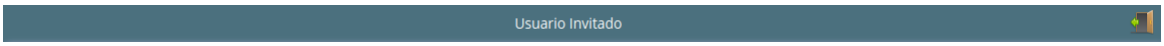
Resultado final con estilos clásicos:



Comenzaremos a realizar los cambios por la parte superior de la pantalla.

Primero se visualizaran los estilos actuales de la nueva versión que los llamaremos “estilos nuevos” y a continuación los estilos de la versión clásica que los llamaremos “estilos clásicos”

Barra superior:



- **Estilos nuevos**

```
.main-top-bar {
    color:#fff;
    background:linear-gradient(#98c6e7,#5376b0);
    background:-o-linear-gradient(#98c6e7,#5376b0);
    background:-moz-linear-gradient(#98c6e7,#5376b0);
    background:linear-gradient(#98c6e7,#5376b0);
    padding: 8px 5px 5px;
   }
```

- **Estilos clásicos**

```
.main-top-bar {
    color:#fff;
    background-color: #4B707E;
    padding: 8px 5px 5px;
   }
```

Barra central:

- **Estilos nuevos**

```
.main-center {
    background:linear-gradient(#404040,#929D9F);
    background:-o-linear-gradient(#404040,#929D9F);
    background:-moz-linear-gradient(#404040,#929D9F);
    background:linear-gradient(#404040,#929D9F);
    box-shadow:3px 3px 8px #000000 inset;
    margin-bottom: 14px;
    padding:20px;
   }

#descrTitle {
    font-size: 18px;
    color: #fff;
   }

#title {
    font-size: 70px;
    color: #fff;
    font-weight: bold;
   }

#version {
```

```
    font-size: 11px;
    color: #fff;
    font-style: italic;
   }
```

- **Estilos clásicos**

```
.main-center {
    background:linear-gradient(#dee9ed,#bed7e1);
    background:-o-linear-gradient(#dee9ed,#bed7e1);
    background:-moz-linear-gradient(#dee9ed,#bed7e1);
    background:linear-gradient(#dee9ed,#bed7e1);
    box-shadow: 3px 0 6px 0 #b4b3dc;
    text-shadow: 0 3px 5px #a1bfbf;
    margin-bottom: 14px;
    padding:20px;
   }

#descrTitle {
    font-size: 18px;
    color: #284550;
   }

#title {
    font-size: 70px;
    color: #284550;
    font-weight: bold;
   }

#version {
    font-size: 11px;
    color: #284550;
    font-style: italic;
   }
```

Módulos:

El módulo lo dividiremos en dos partes: bloque del contenedor y cabecera del contenedor.

Bloque contenedor:

Esta parte hace referencia a la estructura del bloque, aquí es donde se insertará el sombreado del contenedor y el color del borde del modulo.

- **Estilos nuevos**

```
.panel-modules {
    background-color: #e5ecef;
    border: 1px solid #4b707e;
    border-radius: 4px;
    box-shadow: 0px 3px 10px #999999;
   }
```

- **Estilos clásicos**

```
.panel-modules  {
    background-color: #fff;
    border-style: solid;
    border-color: #759DCB;
    border-radius: 4px;
    box-shadow: 0px 3px 10px #999999;
   }
```

Cabecera del contenedor:

- **Estilos nuevos**

```
.main-modules .col-md-4 .title-module {
    background-color: #5093ac;
    box-shadow: 3px 0 6px 0 #b4b3dc;
    color:#fff;
    border-radius: 4px;
    font-size: 14px;
    padding:8px;
    font-weight: bold;
    margin: 2px;
   }
```

- **Estilos clásicos**

```
.main-modules .col-md-4 .title-module {
    background:linear-gradient(#98c6e7,#5376b0);
    background:-o-linear-gradient(#98c6e7,#5376b0);
    background:-moz-linear-gradient(#98c6e7,#5376b0);
    background:linear-gradient(#98c6e7,#5376b0);
    color:#fff;
    border-radius: 4px;
    font-size: 14px;
    padding:8px;
    font-weight: bold;
    margin: 2px;
   }
```

7.5.22.2. Panel FIL

Modificaremos el panel FIL con los estilos actuales para que se parezca lo más posible a los estilos clásicos de las versiones anteriores de gvHidra.

Pantalla actual:

Resultado final con estilos clásicos:

Comenzaremos a realizar los cambios por la parte superior de la pantalla.

Primero se visualizaran los estilos actuales de la nueva versión que los llamaremos “estilos nuevos” y a continuación los estilos de la versión clásica que los llamaremos “estilos clásicos”

Barra superior:

Primero tenemos que cambiar de color el botón del menú que se encuentran en la hoja de estilos layers-menu-cit.css.

- **Estilos nuevos**

```
.horbaritem {
    background:linear-gradient(#a9d5f7,#7ea4d9);
    background:-o-linear-gradient(#a9d5f7,#7ea4d9);
    background:-moz-linear-gradient(#a9d5f7,#7ea4d9);
    background:linear-gradient(#a9d5f7,#7ea4d9);
    border:1px solid;
    border-color:#7593ba;
    border-radius:8px;
    color: #FFFFFF;
    float: left;
    font-family: verdana, arial, helvetica, sans-serif;
    font-size: 12px;
    padding: 2px 10px;
```

```
    white-space: nowrap;
   }

.horbaritem:hover {
    background:linear-gradient(#7ea4d9,#a9d5f7);
    background:-o-linear-gradient(#7ea4d9,#a9d5f7);
    background:-moz-linear-gradient(#7ea4d9,#a9d5f7);
    background:linear-gradient(#7ea4d9,#a9d5f7);
   }
```

- **Estilos clásicos**

```
.horbaritem {
    background-color:#5093AC;
    border:1px solid;
    border-color:#7593ba;
    border-radius:8px;
    color: #FFFFFF;
    float: left;
    font-family: verdana, arial, helvetica, sans-serif;
    font-size: 12px;
    padding: 2px 10px;
    white-space: nowrap;
   }

.horbaritem:hover {
    background-color:#4B707E;
   }
```

Después modificamos el color de la barra:

- **Estilos nuevos**

```
.top-bar {
    background: linear-gradient(#98c6e7,#5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    background: -moz-linear-gradient(#98c6e7, #5376b0);
    background: linear-gradient(#98c6e7,#5376b0);
    color: #fff;
    font-weight: bold;
    font-size: 14px;
    padding: 8px 8px 8px;
   }
```

- **Estilos clásicos**

```
.top-bar{
    background-color: #4B707E;
    color: #fff;
    font-weight: bold;
    font-size: 14px;
    padding: 8px 8px 8px;
   }
```

Barra estado:

Nueva búsqueda

Cambiamos el color de la barra que contiene la botonera que hace referencia al estado del panel.

- **Estilos nuevos**

```
.button-bar {
    .button-bar {
    margin-bottom:10px;
    padding: 10px 10px 10px;
    background: linear-gradient(#606060, #7b7b7b);
    background: -o-linear-gradient(#606060, #7b7b7b);
    background: -moz-linear-gradient(#606060, #7b7b7b);
    background: linear-gradient(#606060, #7b7b7b);
    border: thin solid #606060;
   }
```

- **Estilos clásicos**

```
.button-bar {
    margin-bottom:10px;
    padding: 10px 10px 10px;
    background: #C8DCE5;
    border: thin solid #606060;
   }
```

Barra título del panel:

Ejemplo patron Tabular

- **Estilos nuevos**

```
.topBar-panelFil, .bottomBar-panelFil {
    color: #fff;
    text-align: right;
    padding: 5px;
    background: linear-gradient(#6f91ac,#476286);
    background: linear-gradient(#6f91ac,#476286);
    background: -moz-linear-gradient(#6f91ac, #476286);
    background: linear-gradient(#6f91ac,#476286);
    border-radius: 8px 8px 8px 8px;
   }
```

- **Estilos clásicos**

```
.topBar-panelFil, .bottomBar-panelFil {
    color: #fff;
    text-align: right;
    padding: 5px;
    background-color: #5093AC;
    font-size:13px;
    border-radius: 8px 8px 8px 8px;
   }
```

Capítulo 8. Bitacora de cambios aplicados a gvHidra

8.1. Como migrar mis aplicaciones a otra versión de gvHidra

En este documento apuntaremos las acciones a realizar en una aplicación que utilice IGEP, para pasar de una versión a otra. Cada vez que hagamos alguna modificación en igep que vaya a afectar a las aplicaciones que lo utilicen, lo apuntaremos aqui. Este documento está orientado a Informáticos.

8.1.1. Versión 4.4.22

24-02-2017

- En el menuHerramientas, la entrada a limpiar plantillas cambia

```
    <opcion titulo="Netejar plantilles" descripcion="Netejar plantilles de
smarty" url="phrame.php?action=cleaner">
```

- Repasar tpl's para utilizar las recomendaciones de la rejilla (Guia_DisenyoRejilla_CSS.pdf que se encuentra en Documentos)

8.1.2. Versión 4.4.13

14-09-2016

- Tratamiento de subida de ficheros (fileUpload). El fileUpload de esta versión puede ser múltiple o simple, por ello el tratamiento de la información desde la clase difiere de las versiones 4.2.x.

```
$datosFile = $objDatos->getFileInfo('nomVarFileUpload]');
```

En la variable $mDatosUpload tendremos la siguiente estructura:

```
[nomVarFileUpload] => Array
            (
                [name] => Array
                    (
                        [0] => Fichero1.pdf
                        [1] => Fichero2.pdf
                        ...
                        [n] => FicheroN.pdf
                    )

                [type] => Array
```

```
                    (
                        [0] => application/pdf
                        [1] => application/pdf
                        ...
                        [n] => application/pdf
                    )

                [tmp_name] => Array
                    (
                        [0] => rutaTMP.tmp
                        [1] => rutaTMP.tmp
                        ...
                        [n] => rutaTMP.tmp
                    )

                [error] => Array
                    (
                        [0] => 0
                        [1] => 0
                        ...
                        [n] => 0
                    )

                [size] => Array
                    (
                        [0] => xxxx
                        [1] => xxxx
                        ...
                        [n] => xxxx
                    )
            )
```

- Limpieza en tpl's, es decir, algunos parámetros ya no son necesarios aunque no quitarlos de la tpl no afectan al funcionammiento.

Versiones 4.2.x: {CWLista nombre="ListaMultiple" **multiple="true"** editable="true" visible="true" value=$defaultData_Usuarios.fil_ROL dataType=$dataType_Usuarios.fil_ROL textoAsociado="Lista Múltiple"}
Versiones 4.4.x: {CWLista nombre="ListaMultiple" editable="true" visible="true" value=$defaultData_Usuarios.fil_ROL dataType=$dataType_Usuarios.fil_ROL textoAsociado=#smty_LabelListaMultiple#}

Versiones 4.2.x: {CWFichaEdicion **id="FichaEdicion"** datos=$smty_datosFicha numPagInsertar="1"}
Versiones 4.4.x: {CWFichaEdicion datos=$smty_datosFicha}

Versiones 4.2.x: {CWTabla rowEdit="tabla" conCheck="true" conCheckTodos="false" **id="Tabla1" numPagInsertar="1"** numFilasPantalla="6" datos=$smty_datosTabla}
Versiones 4.4.x: {CWTabla rowEdit="tabla" conCheck="true" conCheckTodos="true" numFilasPantalla="6" datos=$smty_datosTabla}

Versiones 4.2.x: {CWPanel id="edi" **tipoComprobacion="envio"** action="operarBD" method="post" estado="$estado_edi" claseManejadora="pruebaTipos"}
Versiones 4.4.x: {CWPanel id="edi" action="operarBD" method="post" estado="$estado_edi" claseManejadora="pruebaTipos"}

Versiones 4.2.x: {CWBotonTooltip **imagen="01"** iconCSS="glyphicon glyphicon-plus" titulo="Insertar registros" funcion="insertar" actuaSobre="ficha"}
Versiones 4.4.x: {CWBotonTooltip iconCSS="glyphicon glyphicon-plus" titulo="Insertar registros" funcion="insertar" actuaSobre="ficha"}

Versiones 4.2.x: {CWBoton **imagen="41"** iconCSS="glyphicon glyphicon-ok" texto="Guardar" class="button" accion="guardar"}
Versiones 4.4.x: {CWBoton iconCSS="glyphicon glyphicon-ok" texto="Guardar" class="button" accion="guardar"}

8.1.3. Versión 4.2.19

14-09-2016

el acceso a la consola de Debug de la aplicación ha cambiado su url, el acceso desde el xml del menú ahora es de la siguiente forma:

```
<opcion titulo="Log Aplicación" descripcion="Log Aplicación"
url="phrame.php?action=openDebugger" abrirVentana="true" imagen="menu/
mensajes.gif">
```

8.1.4. Versión 4.2.0

17-02-2015

- Etiqueta customTitle pasa a ser la descripción de la aplicación en SESIÓN

```
$sess['daplicacion']
```

- CWBarra -> parametro customTitle "deprecated" se debe utilizar "barTitle" (compatible hacia detrás) En las plantillas se cambiará:

```
{CWBarra usuario=$smty_usuario codigo=$smty_codigo barTitle=
$smty_barTitle}
```

- En esta versión un cambio importante es la ubicación de la carpeta "_Custom_". Hasta ahora se incluía dentro de la carpeta igep, a partir de ahora la ubicación ya puede ser externa a dicha carpeta. El path de ubicación se indicará en un atributo de la etiqueta CustomDirName que se encuentra en el fichero de configuración gvHidraConfig.xml.inc.

- Se han realizado los siguientes cambios en el fichero gvHidraConfig.inc.xml para introducir nuevas funcionalidades y mejoras de uso (entre ellos el indicado en el punto anterior para el custom):

  - Etiqueta startMsg. Puede surgir la necesidad de tener que avisar al usuario de la aplicación sobre una próxima actuación en la aplicación, por ejemplo, un bloqueo o algún cambio próximo. El mensaje puede ser bloqueante, no dejando acceder a la misma, o simplemente informativo.

```
        <!-- initMsg Contenido del mensaje de aranque o ruta al fichero que lo
 contiene    -->
          <!ELEMENT startMsg (#PCDATA)>
          <!ATTLIST startMsg
              id         ID    #REQUIRED
              path    CDATA    #IMPLIED
              type (
                  info | lock
              )
              "info"
          >
```

  - Etiqueta extConfigDir. Puede que ya exista en nuestra DTD, pero ha cambiado su funcionalidad. Facilitará los despliegues, ya que con esta variable se podrá indicar la ruta en la que la aplicación deberá recuperar su configuración. Sustituir la definición anterior por esta:

```
         <!-- extConfigDir Establece el directorio de configuracion externa --
>
          <!ELEMENT extConfigDir (#PCDATA)>
          <!ATTLIST extConfigDir
              fileName CDATA "gvHidraConfig.inc.xml">
```

  - Etiqueta breadcrumb. Con esta etiqueta los usuarios pueden visualizar en una línea de texto el recorido de submenús que se ha seguido y permita regresar a cualquiera de ellos.

```
        <!-- ubicacion Mostrará la ubicación del módulo en el que nos
 encontramos en la pantalla principal -->
          <!ELEMENT breadcrumb EMPTY>
          <!ATTLIST breadcrumb
              visible (false | true) "false">
```

  - Bloque propertyZone. Zona donde podrán definirse entradas de tipo “clave->valor” por el programador. Por ejemplo, para cambiar valores de preproducción a producción.

```
        <!-- propertyZone Seccción que contendrá distintas properties -->
          <!ELEMENT propertyZone (property)*>

          <!-- property
              Se utiliza para definir pares clave (id) valor en el XML de
 forma que
```

```
            puedan cumplimentarse parámetros no contemplados en el XML y
que sean
            necesarios para desarrollar la APP manteniendo la libertad de
modificar
            sus valores desde este fichero de configuración
        -->
        <!ELEMENT property (#PCDATA)>
        <!ATTLIST property
            id          ID    #REQUIRED
        >
```

- Etiquetas para DSN a un Web Service. Se establece una serie de etiquetas para poder definir todos los parámetros que son necesarios para acceder a un web service, certificados y accesos a SAFE.

```
      <!-- wsDSN Define un DSN de conexión a un Web Service. Se
indentificará por un atributo "id" único. -->
        <!ELEMENT wsDSN (uriWSDL?, wsUser?, wsPassword?,
wsCertificatePath?, wsKeyPath?, wsPassPhrase?, wsPAITraceIdApp?) >
        <!ATTLIST wsDSN
            id          ID    #REQUIRED
        >
        <!-- uriWSDL URI para localizar el fichero descriptor del servicio
Web wsdl (Web Service Descriptor Language)  -->
        <!ELEMENT uriWSDL (#PCDATA)>
        <!-- wsUser Si el WS requiere autenticación vía USER/PASS indica el
usuario -->
        <!ELEMENT wsUser (#PCDATA)>
        <!-- wsUser Si el WS requiere autenticación vía USER/PASS indica la
contraseña -->
        <!ELEMENT wsPassword (#PCDATA)>
        <!-- wsCertificatePath Si el WS requiere autenticación vía
certificado, indica localización del fichero -->
        <!ELEMENT wsCertificatePath (#PCDATA)>
        <!ATTLIST wsCertificatePath
            serial CDATA #REQUIRED
            certificateFormat (PEM) "PEM"
        >
        <!-- wsKeyFilePath Si el WS requiere autenticación vía certificado,
indica localización del fichero KEY -->
        <!ELEMENT wsKeyPath (#PCDATA)>
        <!-- wsKeyPath Si el WS requiere autenticación vía certificado,
indica la frase de paso del mismo -->
        <!ELEMENT wsPassPhrase (#PCDATA)>
        <!-- wsPAITraceIdApp Si el WS es de la PAI se indica el ID de la
aplicación para trazabilidad -->
        <!ELEMENT wsPAITraceIdApp (#PCDATA)>
        <!-- enablePAITrace Indica si se activa o no la trazabilidad en la
invocación al WS de la PAI-->
        <!ATTLIST wsPAITraceIdApp
            enablePAITrace (true | false) "true"
        >
```

- Atributo path para indicar la ruta al Custom. El directorio custom, hasta ahora estaba incluido dentro de la carpeta igep, en esta versión el Custom pasará a estar fuera de la carpeta igep. Ventaja: cuando se actualice la versión de gvHIDRA ya no será necesario tener la precaución de salvaguardar el custom de la aplicación. Para ello se añade a la etiqueta CustomDirName el atributo path, con él se indicará la ruta al custom.

```
       <!-- customDirName
              Establece el nombre del directorio de customización. Son
modificaciones
              o extensiones    que van a ser comunes a una organización o a
varias
              aplicaciones. Por ejemplo, las extensiones del Framework
propias de la CIT
          -->
          <!ELEMENT customDirName (#PCDATA)>
          <!ATTLIST customDirName
              path    CDATA    #IMPLIED
          >
```

- Hasta ahora teníamos el custom en la siguiente ruta: igep/custom/nombreCustom

  Ahora, lo tendremos al mismo nivel que la carpeta igep: custom/nombreCustom

De los ficheros css que se han creado, es interesante añadir loading.css: ya que mejora mucho el "cargando", debugger.css, y login.css Los estilos añadidos al fichero aplicacion.css solamente es obligatorio definir el .background ya que es el estilo que cargará por defecto si no utilizamos .backgroundFil, .backgroundLis o backgroundEdi.

8.1.5. Versión 4.0.x

15-04-2013

Aunque no es un cambio obligatorio de migración, en los custom que se publican, se ha dividido el fichero aplicacion.css en varios para tener mejor organizados los estilos, todos ellos se importan desde aplicacion.css que sigue siendo el principal.

Los ficheros son los siguientes:

- loading.css: Contiene un nuevo cargando, mejora bastante el que teníamos antes.
- debugger.css: Contiene los estilos que se aplicarán al debugger.
- login.css: Estilos para la pantalla de validación, se incorpora que aparezca en esta pantalla el nombre de la aplicación y el número de versión.
- bumpbox.css: Estilos correspondientes a la funcionalidad bumpbox.
- tree.css: Estilos correspondientes al patrón árbol.
- aplicacion.css: Se han añadido 5 estilos nuevos que son:

  .frame: Estilo que se puede aplicar al marco que engloba la pantalla, plugin CWMarcoPanel.

.panel: Estilo que se puede aplicar al panel (fil, lis o edi), plugin CWPanel.

.background: Este estilo hay que añadirlo para que sea el de por defecto sino se especifica .background-Fil .backgroundEdi o .backgroundLis

.goToPage: Estilo para el campo que aparece al lado del paginador donde se introducirá la página a la que se quiere acceder directamente.

.columnTitle img: Estilo que se puede aplicar a las imágenes de ordenación que aparecen en los títulos de las columnas de un tabular.

#detalles: Es una capa que engloba las solapas de los detalles, en el caso de que haya más de uno.

De los ficheros css que se han creado, es interesante añadir loading.css: ya que mejora mucho el "cargando", debugger.css, y login.css Los estilos añadidos al fichero aplicacion.css solamente es obligatorio definir el .background ya que es el estilo que cargará por defecto si no utilizamos .backgroundFil, .backgroundLis o backgroundEdi.

8.1.6. Versión 4.0.0

24-01-2013

- Fijar el parametro templatesCompilationDir en el xml. En el custom de la cit tenemos que quitar lo que fija este parametro para que todos pasen a hacerlo.
- Incluir en el fichero gvHidraConfig.inc.xml de la aplicación modificar la DTD:
  - En la cabecera:

```
<!ELEMENT gvHidraConfig (
gvHidraVersion |
applicationName |
appVersion |
customTitle |
customDirName |
templatesCompilationDir |
reloadMappings |
smartyCompileCheck |
logSettings |
queryMode |
extConfigDir | smtpServer |
DSNZone
)
```

  - En la definición de elementos añadimos:

```
  <!-- extConfigDir
            Establece el directorio de configuracion externa
       -->
       <!ELEMENT extConfigDir (#PCDATA)>
```

```
<!-- smtpServer
      Define el servidor de SMTP
   -->
   <!ELEMENT smtpServer (smtpHost, smtpPort? , smtpUser?, smtpPassword?)>

   <!ELEMENT smtpHost (#PCDATA)>
   <!ELEMENT smtpPort (#PCDATA)>
   <!ELEMENT smtpUser (#PCDATA)>
   <!ELEMENT smtpPassword (#PCDATA)>
```

- Añadimos en el elemento dbDSN la posibilidad de gestionar SQL Server:

```
<!ELEMENT dbDSN (dbHost, dbPort?, dbDatabase?, dbUser, dbPassword)? >
      <!ATTLIST dbDSN
          id             ID        #REQUIRED
          sgbd (
              postgres | pgsql |
              oracle | oci8 | oci |
              thin | oracle-thin |
              mysql | sqlite | sqlsrv
          ) "postgres"
>
```

- Sustituir en todos los ficheros TPLs: ' class="boton" ' por ' class="button" '
- Cambiar el parámetro datos del CWLista por value.
- Cambiar el parámetro valor del CWCheckbox por value.
- Cambiar el parámetro valor del CWSelector por value.
- Cambiar el metodo showMensaje(por showMessage(
- Cambiar el metodo setPametrosBusqueda(por setSearchParameters(
- Cambiar addAccionInterfaz(por addTriggerEvent(
- Cambiar addPadre(por addMaster(
- Cambiar addHijo(por addSlave(
- Cambiar setParametrosBusqueda(por setSearchParameters(
- Cambiar setLimiteConsulta(por setLimit(
- Cambiar addConstante(por addConstant(
- Cambiar setTipoConsulta(por setQueryMode(
- Cambiar getTipoConsulta(por getQueryMode(
- Cambiar setResultadoBusqueda(por setResultForSearch(
- Cambiar getResultadoBusqueda(por getResultForSearch(

- Cambiar setResultadoEdicion(por setResultForEdit(
- Cambiar getResultadoEdicion(por getResultForEdit(
- Cambiar setLista(por setList(
- Cambiar getLista(por getList(
- Cambiar getModoActivo(por getActiveMode(
- Cambiar setOperacion(por setOperation(
- Cambiar getOperacion(por getOperation(
- Cambiar setIndice(por setIndex(
- Cambiar getIndice(por getIndex(
- Cambiar hayDatos(por isEmpty(. ATENCION: este cambio cambia el valor devuelto por el método. Ahora, tendremos que negar el valor recibido para obtener el comportamiento que teniamos hasta ahora. Es decir, para obtener si hay datos será !isEmpty.
- Si se dispone de sistema de validación propio (es decir, se ha creado una clase que extiende la clase gvhBaseAuth), el método fetchData tiene un parámetro más. Ahora se queda: function fetchData($username, $password, $isChallengeResponse=false)
- Cambio en las CSS, la migración del css del custom se debe hacer siguiendo el pdf adjunto (migracionCSSgvHidra4.pdf) En las aplicaciones hay que modificar los estilos de todas las TPLs, principalmente:
  - "formularios" por "text"
  - "boton" por "button"

8.1.7. Versión 3.1.0

23-11-2010

- Si se está utilizando de forma complementaria al framework el proyecto jasper para creación de listados, en esta versión ya se puede utilizar la versión 'jasper-3_0_0_4'.
- Si el método **IgepComunicacion::setAllTuplas** no recibe un array corta la ejecución. Asegurarse que si se le pasa la salida del método consultar, hayan registros.
- Si se ha definido algún método nuevo de autenticación, en método autenticate añadir **IgepSession::session_start($p_apli, false);** antes de crear instancia de Auth.
- Si se usan clientes de web services con la clase **IgepWS_Client** y no se ha fijado la opción de codificación, ahora se fija a **latin1**. Con esto ya no es necesario usar los métodos uff8_encode/utf8_decode para convertir los datos.
- **(RECOMENDADO)** El atributo **logSettings** ya no se fija en la carga dinámica; añadirlo al **gvHidraConfig.inc.xml** según el nivel de log deseado en cada servidor donde se despliegue la aplicación.
- **(RECOMENDADO)** Si se usan fechas en servidores de web services se recomienda generarlas usando el método formatSOAP de **gvHidraTimestamp**.

- Cambiamos el nombre del método addCamposClave por **setPKForQueries**.
- El método ConfigFramework::setCustomDirName ya no existe. Si se ha definido algún custom nuevo, hay que actualizar la DTD porque ya no se puede fijar el atributo **customDirName** (en los xml de la aplicación y del framework si se puede). Podéis copiar la DTD correcta del xml de cualquier custom.
- Los métodos de **ConfIgep** *formatoFecha, formatoNumero y formatoFechaNegocio* ya no existen.
- Los métodos de **IgepComunicaUsuario** *strtotime_es y timetostr_es* ya no existen.
- El método *ConfIgep::es_desarrollo* ya no existe. Si hay que hacer alguna acción dependiente del servidor, intentar configurar el xml para cada servidor, o si no hay más remedio usar el nombre del host para decidir. Cuestiones relacionadas:
  - En el servidor de producción ya no se fija el error_reporting (E_USER_ERROR | E_USER_WARNING | E_USER_NOTICE). Se hará como esté definido en el servidor PHP.
  - Fijar el parámetro **<reloadMappings>false</reloadMappings>** en **producción**, ya que por defecto está habilitado.
  - Fijar el parámetro **<smartyCompileCheck>false</smartyCompileCheck>** en **producción**, ya que por defecto está habilitado.
- Actualizar la **DTD** del fichero **gvHidraConfig.inc.xml**. Podéis copiar la DTD correcta del xml de la plantilla de aplicación. Cambios realizados:
  - Se fija el rango del atributo status de queryMode de 0 a 2. Si se ha fijado este elemento comprobar que es el deseado. El valor 3 ya no existe por lo que si se estaba usando (en ConfigFramework::setQueryMode, gvHidraForm_DB::setQueryMode o gvHidraSelectionWindow::setQueryMode) cambiar por valor 1.
  - Cambia el atributo dnsRef de logSettings por el atributo dsnRef.
  - Se añade el elemento opcional temporalDir, usado para fijar la ubicación del fichero de sesión. (CIT: Es imprescindible para uso en producción)
- Cambios en las listas. La clase IgepLista pasa a llamarse **gvHidraList**, podemos definir una fuente de datos externa. Pasos de migración obligatorios:
  - Reemplazar en todo el código cualquier ocurrencia de *"IgepLista" -> "gvHidraList"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"$this->addLista" por "$this->addList"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->marcarSeleccionado" por "->setSelected"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->setSeleccionado" por "->setSelected"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->addOpcion" por "->addOption"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->deleteOpcion" por "->deleteOption"*.
  - Reemplazar en todo el código cualquier ocurrencia de *"->setDefList" por "->setList_DBSource"*.
- Cambios en las ventanas de selección. La clase IgepVentanaSeleccion pasa a llamarse **gvHidraSelectionWindow**, todas las ventanas de selección tienen matching, podemos definir fuentes de datos externas, podemos fijar una tpl a partir de la cual mostrar dicha ventana.

- Reemplazar en todo el código cualquier ocurrencia de *"IgepVentanaSeleccion" por "gvHidraSelectionWindow"*.

- Reemplazar en todo el código cualquier ocurrencia de *"$this->addVentanaSeleccion" por "$this->addSelectionWindow"*.

- *Añadir addMatching*. Para poder reutilizar las ventanas de selección en diferentes paneles, ahora incorporan Matching. Esto quiere decir, que tendrán un nombre los campos resultado y nosotros podremos asociarlos a nuestros campos de la tpl. A efectos de migración, esto supone que, el tercer parámetro de las antiguas llamadas a IgepVentanaSeleccion se transforma en una o varias llamadas al método addMatching.

```
// VERSIÓN ANTERIOR
$codper = new IgepVentanaSeleccion("codper","PERSONAS",array("codper","nom"));
$this->addVentanaSeleccion($codper);

// VERSIÓN 3.1.1
$codper = new gvHidraSelectionWindow("codper","PERSONAS"); //Eliminamos el
 array.
$codper->addMatching('codper','codper'); //Creamos tantas llamadas al metodo
 addMatching como elementos del array
$codper->addMatching('nom','nom');  // Nota: los dos parametros serán el mismo
 para que funcione como en la version 3.0.X
$this->addSelectionWindow($codper);
```

- Reemplazar en todo el código cualquier ocurrencia de *"->setDefVS" por "->setSelectionWindow_DBSource"*.

- Reemplazar en todo el código cualquier ocurrencia de **"->setDependencia" por "->setDependence"**.

- Cambio del nombre de la clase de cheks. Reemplazar en todo el código cualquier ocurrencia de "IgepCheckBox" por "**gvHidraCheckBox**"

Parte V. Apendices

Tabla de contenidos

A. FAQs, resolución de problemas comunes 253
A.1. ¿Cómo puedo recargar el maestro desde el detalle? 253
A.2. ¿Qué opciones de impresión/exportación ofrece el framework? 253
B. Pluggins, que son y como usarlos en mi aplicación 254
B.1. Documentación Plugins gvHidra 254
B.1.1. CWArbol 255
B.1.2. CWAreaTexto 256
B.1.3. CWBarra 257
B.1.4. CWBarraInfPanel 258
B.1.5. CWBarraSupPanel 259
B.1.6. CWBoton 259
B.1.7. CWBotonTooltip 262
B.1.8. CWCampoTexto 265
B.1.9. CWContendorPestanyas 267
B.1.10. CWContenedor 268
B.1.11. CWCheckBox 269
B.1.12. CWFicha 270
B.1.13. CWFichaEdicion 271
B.1.14. CWFila 272
B.1.15. CWImagen 273
B.1.16. CWLabel 274
B.1.17. CWMaps 276
B.1.18. CWInfoContenedor 277
B.1.19. CWInformation 277
B.1.20. CWLista 278
B.1.21. CWMarcoPanel 280
B.1.22. CWMenuLayer 281
B.1.23. CWPaginador 283
B.1.24. CWPanel 283
B.1.25. CWPantallaEntrada 286
B.1.26. CWPestanyas 286
B.1.27. CWRichAreaTexto 287
B.1.28. CWSelector 289
B.1.29. CWSlider 290
B.1.30. CWSolapa 291
B.1.31. CWTabla 292
B.1.32. CWTree 293
B.1.33. CWUpLoad 294
B.1.34. CWVentana 295
C. Listados Jasper en gvHIDRA 297
C.1. Integracion de Jasper en un mantenimiento 297
C.2. Depuración de errores al intentar ejecutar informes JasperReports en gvHidra 300
C.3. Recomendaciones y generalidades sobre integración gvHidra y Jasper 302
D. Instalación rápida del entorno 305
D.1. Introducción 305
D.2. Requisitos 305
D.3. Instalación entorno rápido 305

Apéndice A. FAQs, resolución de problemas comunes

A.1. ¿Cómo puedo recargar el maestro desde el detalle?

Puede que, en algunas ventanas, sea necesario recargar los datos del maestro tras modificar el detalle. Para ello, el framework desde su versión 3.1.4 proporciona el método refreshMaster que se podrá invocar desde los detalles.

```
$this->refreshMaster();
```

En versiones anteriores del framework esto se debía hacer con un parche. Las instrucciones son:

```
//Recuperamos el panel maestro
$master=IgepSession::damePanel('ClaseMaestro');
$master->regenerarInstancia();
//Compartimos conexion. De ese modo todos los campos pendientes se
 visualizaran en el maestro.
$master->obj_conexion = $this->obj_conexion;
//Recargamos el contenido sin recargar los detalles
$master->refreshSearch(false);
IgepSession::guardaPanel('ClaseMaestro',$master);
```

A.2. ¿Qué opciones de impresión/exportación ofrece el framework?

Dependiendo de lo que se requiera, gvHIDRA ofrece una serie de posibilidades de impresión/exportación. La más sencilla es utilizar la acción genérica de impresión. Para ello debemos incluir en nuestro panel un botontooltip con la acción print y el panel sobre el que actua (tabla o ficha).

```
{CWBotonTooltip imagen="06" titulo="Imprimir" funcion="print"
 actuaSobre="tabla"}
```

Otra de las opciones genéricas es la exportación a CSV. En este caso, la acción es exportCSV y se le tiene que pasar también el panel sobre el que se actua (tabla o ficha).

```
{CWBotonTooltip imagen="csv" titulo="Exportar a csv" funcion="exportCSV"
 actuaSobre="tabla"}
```

Para realizar listados o informes más complejos, se puede hacer uso de proyectos como jasperReport (con la librería jasper de gvHIDRA) o PHPOdt. Para los primeros, consultar este manual en la sección adecuada.

Apéndice B. Pluggins, que son y como usarlos en mi aplicación

B.1. Documentación Plugins gvHidra

Lista de Pluggins

1. CWArbol [255]
2. CWAreaTexto [256]
3. CWBarra [257]
4. CWBarraInfPanel [258]
5. CWBarraSupPanel [259]
6. CWBoton [259]
7. CWBotonTooltip [262]
8. CWCampoTexto [265]
9. CWContendorPestanyas [267]
10. CWContenedor [268]
11. CWCheckBox [269]
12. CWFicha [270]
13. CWFichaEdicion [271]
14. CWFila [272]
15. CWImagen [273]
16. CWLabel [274]
17. CWMaps [276]
18. CWInfoContendor [277]
19. CWInformation [277]
20. CWLista [278]
21. CWMarcoPanel [280]
22. CWMenuLayer [281]
23. CWPaginador [283]
24. CWPanel [283]
25. CWPantallaEntrada [286]
26. CWPestanyas [286]

27.CWSelector [289]

28.CWSlider [290]

29.CWSolapa [291]

30.CWTabla [292]

31.CWTree [292]

32.CWUpLoad [294]

33.CWVentana [295]

B.1.1. CWArbol

Crea paneles vinculados a una estructura jerárquica.

En la parte izquierda dibuja el árbol, y el panel de la derecha corresponde al nodo seleccionado.

- **Plugins que pueden contener a CWArbol**
  - Padres
    - CWMarcoPanel
- **Plugins que puede contener CWArbol**
  - Hijos
    - CWPanel

Tabla de argumentos de CWArbol

Nombre	Tipo	¿Opcional?	Descripción	Valores
estado	alfanumérico	false	Indica el estado del panel (activo o desactivado) cuando se carga la pantalla. Normalmente es una variable smarty ($estado_edi) que será sustituida por el php correspondiente del views.	Puede tomar los siguientes valores: • *on*: visible y activo • *off*: visible e inactivo • *inactivo*: no visible e inactivo
arbol	alfanumérico	false	Estructura definida para el árbol (fichero xml). Será una variable smarty ($smty_objArbol) que es sustituida por IgepPanel.	
titulo	alfanumérico	true	Dará el título al panel que contiene el árbol.	
ancho	numerico	true	Define el ancho del panel árbol en porcentaje, por defecto es un 25%.	

Ejemplos de uso del plugin CWArbol:

Árbol con dos paneles diferentes. La variable **$smty_panelVisible** es sustituida por **IgepPanelArbol**, que le asignará el valor del tipo de nodo que se ha seleccionado, coincidrá con el nombre que se le pase como primer parámetro a la función **setNodoPanel()**.

```
{CWArbol estado=$estado_edi arbol=$smty_objArbol}
 {if $smty_panelVisible == "ANYO" || $smty_panelVisible == "FACTURA"}
   {CWPanel
                    ..............
                {/CWPanel}
 {else}
                {CWPanel
                    ..............
                {/CWPanel}
 {/if}
{/CWArbol}
```

Lista de plugins [254]

B.1.2. CWAreaTexto

Equivalente al TEXTAREA de HTML.

- **Plugins que pueden contener a CWAreaTexto**
  - Padres
    - CWContenedor
    - CWFila
    - CWFicha
    - CWSelector
    - CWSolapa
- **Plugins que puede contener CWAreaTexto**
  - Hijos
    - El plugin CWAreaTexto es una hoja (no contiene otros plugins)

Tabla de argumentos de CWAreaTexto

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
longitudMinima	entero	true	Indica el número de carácteres mínimos que debe introducirse en el campo. Si no se alcanza el número mínimo de caracteres a introducir, se muestra un mensaje de ALERTA.
longitudMaxima	entero	true	Indica el número de carácteres máximos que pueden introducirse en el campo. Si se sobrepasa dicho valor, se muestra un mensaje de ALERTA.
value	alfanumérico	true	Valor por defecto que aparecera en el campo.

Nombre	Tipo	¿Opcional?	Descripción
editable	alfanumérico	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWAreaTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
cols	entero	true	Especifica el número de columnas que tendrá el textarea. En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
rows	entero	true	Especifica el número de filas que tendrá el textarea.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
actualizaA	alfanumérico	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.
class	string	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.

Ejemplos de uso del plugin CWAreaTexto:

Ejemplo: Declaración de CWAreaTexto.

```
{CWAreaTexto nombre="comentario" textoAsociado="Expedientes" cols="20"
 rows="10"}
```

Lista de plugins [254]

B.1.3. CWBarra

Dibuja la barra superior común a todas las ventanas, comprende el menú de la aplicación, una descripción de la pantalla o formulario donde nos encontramos (la entrada de menú correspondiente). El usuario que se ha validado en la aplicación, y por último la hora y fecha local del PC desde el que se ejecuta el navegador Web.

- **Plugins que pueden contener a CWBarra**
  - Padres
    - CWVentana
- **Plugins que puede contener CWBarra**
  - Hijos
    - CWMenuLayer

Tabla de argumentos de CWBarra

Nombre	Tipo	¿Opcional?	Descripción
usuario	alfanumérico	true	Fija el nombre de usuario en la Barra. Variable smarty ($smty_usuario) asignada internamente.
codigo	alfanumérico	true	Fija el nombre de la aplicacion en la barra Variable smarty ($smty_codigo) asignada internamente.
customTitle	alfanumérico	true	En la barra superior de color azul, a la izquierda de la fecha y hora, se ha reservado un pequeño espacio para poder incluir un texto personalizado. La asiganción se realiza a través de la variable smarty ($smty_customTitle) asignada internamente por gvHidra en IgepPantalla
iconOut	alfanumérico	true	Icono que aparecerá a la derecha de la barra superior para salir de la aplicación.
iconHome	alfanumérico	true	Icono que aparecerá a la derecha de la barra superior para acceder a la pantalla principal de la aplicación.
iconInfo	alfanumérico	true	Icono que aparecerá a la derecha de la barra superior que mostrará una ventana de información sobre la aplicación.

Ejemplos de uso del plugin CWBarra:

Ejemplo: Una barra con Menú (plugin **CWMenuLayer**).

```
{CWBarra usuario="$smty_usuario" codigo="$smty_codigo" customTitle=
$smty_customTitle}
  {CWMenuLayer name="$smty_nombre" fichero="$smty_fichero"}
{/CWBarra}
```

Lista de plugins [254]

B.1.4. CWBarraInfPanel

Dibuja la barra inferior de un panel. Contiene los botones que realizan acciones sobre la capa de persistencia (botones inferiores según la guía de estilo).

- **Plugins que pueden contener a CWBarraInfPanel**
  - Padres
    - CWPanel
- **Plugins que puede contener CWBarraInfPanel**

- Hijos
  - CWBoton

Tabla de argumentos de CWBarraInfPanel

El plugin CWBarraInfPanel no tiene argumentos

Ejemplos de uso del plugin CWBarraInfPanel:

Ejemplo: Declaración de CWBarraInfPanel.

```
{CWBarraInfPanel}
  {CWBoton imagen="41" texto="Guardar" class="boton" funcion="guardar"}
{/CWBarraInfPanel}
```

Lista de plugins [254]

B.1.5. CWBarraSupPanel

Dibuja lo que en un panel será la cabecera de la ventana donde irá el título de esa ventana.

- **Plugins que pueden contener a CWBarraSupPanel**
  - Padres
    - CWPanel
- **Plugins que puede contener CWBarraSupPanel**
  - Hijos
    - CWBotonTooltip

Tabla de argumentos de CWBarraSupPanel

Nombre	Tipo	¿Opcional?	Descripción
titulo	alfanumérico	true	Establece la descripción de la acción del panel.

Ejemplos de uso del plugin CWBarraSupPanel:

Ejemplo: Declaración de CWBarraSupPanel.

```
{CWBarraSupPanel titulo="BUSCAR ESTADOS"}
  {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar"}
{/CWBarraSupPanel}
```

Lista de plugins [254]

B.1.6. CWBoton

Equivalente al BUTTON de HTML. Generalmente, el uso de estos botones será para acceso a la capa de persistencia.

- **Plugins que pueden contener a CWBoton**
  - Padres

- CWBarraInfPanel

- **Plugins que puede contener CWBoton**
  - Hijos
    - El plugin CWBoton es una hoja (no contiene otros plugins)

Tabla de argumentos de CWBoton

Nombre	Tipo	¿Opcional?	Descripción	Valores
name	alfanumérico	true	Nombre para identificar la instancia del componente, y se utilizara como texto del botón.	
class	alfanumérico	true	Indica el estilo css del botón que se genera. Por defecto tiene el marcado por la guía de estilo ('boton'). En caso de querer cambiarlo se debería incluir en la css.	
textoAsociado	alfanumérico	true	Fija el texto descriptivo que acompaña a un campo (etiqueta).	
editable	boolean	true	Puede interesarnos inhabilitar un botón para determinados perfiles de usuario o datos.	
mostrarTextoAsociado	boolean	true	No siempre queremos que se muestre la etiqueta o texto asociado, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario.	
iconCSS	alfanumérico	true	Corresponde al selector del icono que queremos, correspondiente a uno de los selectores de las librerías que se tengan cargadas. iconCSS = "glyphicon glyphicon-refresh" -> Librería Glyphicons iconCSS = "fa fa-check" -> Librería Font-Awesome	
checkForm	booleano	true	Con este parámetro indicaremos si queremos que se valide el formulario antes de realizar la ejecución del botón o no. Por defecto siempre se harán las comprobaciones, a no ser que se indique checkForm = "false". Este parámetro solamente tiene efecto en botones con los siguientes valores en accion = ['particular','saltar','volver']	
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHidra.	

Nombre	Tipo	¿Opcional?	Descripción	Valores
editable	enumerado	true	Especifica el comportamiento del campo: si su valor es true, el botón estará activo. Si el valor es false, el botón estará desactivado, y, si su valor es nuevo, el botón solamente se activará en la inserción cuando el plugin CWBotonTooltip sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el botón actúa como editable=true.	
funcion	alfanumérico	true	Este parametro se utiliza por si queremos añadirle funcionalidad extra al botón. En caso de utilizarse no debe añadirse la palabra reservada "javascript:".	
accion	alfanumérico	true	Especifica la acción que realiza el botón	Puede tomar los siguientes valo • *guardar*: Ejecuta la acción d panel. • *cancelar*:Cancela la acción d panel. • *saltar*: La acción es un salto. • *volver*: Se retorna de un salto • *listar*: La accion invoca a un tado. • *cancelarvs*: Para un botón Ca lar de una ventana emergente • *aceptarvs*: Sólo para el botón Aceptar de la ventana de sele ción. • *particular*: El botón realiza u función que no es genérica. E este caso son obligatorios los rámetros "*id*" y "*action*". "a*ction*": acción particular qu ejecutará en la clase manejad (método accionesParticulares "id": identificador del botón. queremos que el botón funci en cuanto a visibilidad como guardar/cancelar pondremos parámetro visible = "false".
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea	

Nombre	Tipo	¿Opcional?	Descripción	Valores
			visible/invisible desde el principio, y tratar su visibilidad posteriormente desde negocio.	
id	alfanumérico	true	Identificador para el botón, sólo en el caso de que la acción sea "particular".	
openWindow	boolean	true	Indica si queremos que la acción del botón la ejecute en una ventana diferente.	
action	alfanumérico	true	Acción que queremos que se ejecute al pulsar el botón y que sea diferente a la de por defecto del panel.	
texto	alfanumérico	true	Este parametro se utiliza para visualizar la ayuda en linea al situar el puntero del ratón sobre el componente.	
confirm	boolean	true	Con este parámetro aparecerá una ventana emergente de confirmación para ejecutar la acción del panel.	
filaActual	entero	true	Nos indica la fila seleccionada, **SÓLO** se utiliza en la **VENTANA DE SELECCIÓN** donde es OBLIGATORIO.	
formActua	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica el nombre del formulario en el que se devolverán los valores elegidos en la ventana de selección.	
panelActua	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica el nombre del panel en el que se devolverán los valores elegidos en la ventana de selección.	
actuaSobre	alfanumérico	true	Parámetro OBLIGATORIO **SÓLO** cuando es un botón de la **VENTANA DE SELECCIÓN**. Indica los campos donde se volcarán los valores elegidos en la ventana de selección.	

Ejemplos de uso del plugin CWBoton:

Ejemplo: Declaración de CWBoton.

```
{CWBoton imagen="41" texto="Guardar" class="boton" accion="guardar"}
```

Lista de plugins [254]

B.1.7. CWBotonTooltip

Estos componentes se utlizan para comunicar la capa de presentación con la capa de la lógica de negocio. Las acciones realizadas a través de ellos no persisten hasta que se confirman con los botones inferiores.

- **Plugins que pueden contener a CWBotonTooltip**
  - Padres
    - CWBarraSupPanel
    - CWFicha
    - CWFila
    - CWSelector
    - CWSolapa
- **Plugins que puede contener CWBotonTooltip**
  - Hijos
    - El plugin CWBotonTooltip es una hoja (no contiene otros plugins)

Tabla de argumentos de CWBotonTooltip

Nombre	Tipo	¿Opcional?	Descripción	Valores
titulo	alfanumérico	false	Nombre para identificar la instancia del componente. Este debe coincidir con la acción que se quiera realizar.	
id	alfanumérico	true	Nombre que identificar el botón. Debe coincidir con la acción que se quiera realizar.	En el caso de que sea un botón tooltip con función *openDoc* el id debe ser el nombre del campo de la consulta que contendrá la ruta al documento.
actuaSobre	alfanumérico	false	En el caso de la actualización de campos aqui se indicará el array de los campos a actualizar. En el resto de casos, indica sobre que panel se va a mostrar el resultado.	Indicará el destino de la acción del botón: • *tabla*: si los datos se muestran sobre una tabla y las operaciones se realizan sobre ella • *ficha*: si los datos se muestran sobre una ficha y las operaciones se realizan sobre ella
mostrarEspera	boolean	true	Su valor por defecto es false. Si la accion que desencadena el boton requiere un tiempo de proceso considerable, es conveniente utilizar este parámetro fijándolo a 'true'. Con ello conseguimos que durante el procesado de la acción que desencadena el botón, la interfaz permanezca bloqueda y se muestre al usuario un mensaje que indica que espere.	
iconCSS	alfanumérico	true	Corresponde al selector del icono que queremos, correspondiente a	

Nombre	Tipo	¿Opcional?	Descripción	Valores
			uno de los selectores de las librerías que se tengan cargadas. iconCSS = "glyphicon glyphicon-refresh" -> Librería Glyphicons iconCSS = "fa fa-check" -> Librería Font-Awesome	
action	alfanumérico	true	Se utiliza para redireccionar en el mapping, la lógica de negocio. **IMPORTANTE:** Botón insertar en panel de búsqueda action siempre valdrá "nuevo".	
funcion	enumerado	true	Indica la función que va a tener asociada ese BotonTooltip (es decir, el tipo de botón que es)	Acción que realizará el botón: • *insertar*: Un botonTooltip de inserción o nuevo • *modificar*: BotonTooltip de edición o modificación • *eliminar*: BotonToltip de marcado para borrar • *limpiar*: BotonToltip de restablecimiento de valores • *abrirVS*: Botón que abre una Ventana de selección • *buscarVS*: Botón que ejecuta la búsqueda dentro de la ventana de selección • *actualizaCampos*: Botón que actualizará una serie de campos • *print*: imprimirá lo visible en la ventana actual. • *exportCSV*: exporta a CSV el contenido del panel • *ayuda*: Botón que nos abrirá el manual de ayuda en una ventana emergente. • *openDoc*: Botón que nos permitirá abrir un documento pasándole una ruta que vendrá dada de BD. Para ello hay que tener en cuenta que el valor del parámetro id de-

Nombre	Tipo	¿Opcional?	Descripción	Valores
				be ser el nombre del campo correspondiente.
filaActual	alfanumérico	true	Parámetro **SÓLO** para el botón buscar de la **VENTANA DE SELECCIÓN** que será una variable smarty de gestión interna.	
claseManejadora	alfanumérico	true	Nombre de la clase php que maneje el panel.	
rutaManual	alfanumérico	false	Indicaremos la ruta relativa a partir del directorio doc, a la página del manual que queremos que se muestre. Obligatorio si el botón es un enlace al manual.	

Ejemplos de uso del plugin CWBotonTooltip:

Ejemplo: Botón de eliminar que se encuentra en un panel tabular e insertará en él mismo.

```
{CWBotonTooltip imagen="01" titulo="Eliminar registros" funcion="eliminar"
 actuaSobre="tabla"}
```

Ejemplo: Botón para inserción en un patrón Tabular-Registro, la inserción se efectuará en el panel registro (ficha) y hay que indicarle un action, con "nuevo" que será la redirección indicada en el mappings.php.

```
{CWBotonTooltip imagen="01" titulo="Insertar registros" funcion="insertar"
 actuaSobre="ficha" action="nuevo"}

$this->_AddMapping('claseManejadora__nuevo', 'claseManejadora');
$this->_AddForward('claseManejadora__nuevo', 'gvHidraSuccess', 'index.php?
view=views/patronesSimples/p_tabularRegistro.php&panel=editar');
$this->_AddForward('claseManejadora__nuevo', 'gvHidraError', 'index.php?
view=views/patronesSimples/p_tabularRegistro.php&panel=listar');
```

Ejemplo: Botón asociado a una ventana de selección.

```
{CWBotonTooltip imagen="13" titulo="solucionar"
 actuaSobre='autor,fsolucion,l_estado' funcion="actualizaCampos"}
```

Lista de plugins [254]

B.1.8. CWCampoTexto

Equivalente al INPUT de HTML.

- **Plugins que pueden contener a CWCampoTexto**
  - Padres
    - CWContenedor
    - CWFila

- CWFicha
- CWSelector
- CWSolapa

- **Plugins que puede contener CWCampoTexto**
  - Hijos
    - El plugin CWCampoTexto es una hoja (no contiene otros plugins)

Tabla de argumentos de CWCampoTexto

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá el tipo de dato a mostrar en el campo (cadena, número, fecha...) y sus propiedades como: longitud, obligatorio, máscara, calendario... Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
autofocus	boolean	true	El campo que tenga este parámetro a true, será el que tenga el foco nada más entrar en la pantalla.
editable	enumerado	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWCampoTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
oculto	boolean	true	Añadido a partir de la versión 1.20. Se utiliza para crear CWCampoTextos no visibles, pero que pueden ser útiles para campos calculados, etc... Su comportamiento es similar al de un CWCampoTexto, a escepcion de la parte visual (CSSs) y el javascript de control.
size	alfanumérico	true	Permite indicar el tamaño del campo en caracteres a la hora de visualizarlo en pantalla. En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
conUrl	boolean	true	Si aparece y su valor es true, al lado de la caja de texto, aparece un botón, que al pulsarse abrirá una nueva ventana del navegador, cargando la URL que tenga como valor el CampoTexto. Deberá ser una URL válida y completa.
value	alfanumérico	true	Valor que queremos que tenga el campo cuando estamos en una inserción.

Nombre	Tipo	¿Opcional?	Descripción
actualizaA	alfanumérico	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	boolean	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
funcion	alfanumérico	false	Llamada a distintas funciones con sus <COMPLETAR>
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el campo sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHidra.
autocomplete	numérico	true	Indica si se quiere que el sistema autocomplete el contenido a través de JQuery. El framework consultará a partir del mappings correspondiente a este campo. El valor introducido indica el caracter a partir del cual el sistema realizará la consulta.
class	string	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.

Ejemplos de uso del plugin CWCampoTexto:

Ejemplo de uso del atributo máscara:

```
{CWCampoTexto nombre="ediCif" size="9" editable="true"
 textoAsociado="CIF" dataType=$smty_dataType_Personas.ediCif}
```

Lista de plugins [254]

B.1.9. CWContendorPestanyas

Plugin que alberga dentro las pestañas asociadas a un panel.

- **Plugins que pueden contener a CWContendorPestanyas**
  - Padres
    - CWMarcoPanel
- **Plugins que puede contener CWContendorPestanyas**
  - Hijos
    - CWPestanya

Tabla de argumentos de CWContendorPestanyas

Nombre	Tipo	¿Opcional?	Descripción
id	alfanumérico	false	Establece el identificador del CWContenedorPestanyas, y el nombre del objeto JS que controla las pestañas

Ejemplos de uso del plugin CWContendorPestanyas:

Ejemplo: Declaración de un CWContenedorPestanya con dos pestanyas dentro.

```
{CWContenedorPestanyas id="Maestro"}
  {CWPestanya tipo="fil" estado=$estado_fil}
  {CWPestanya tipo="lis" estado=$estado_lis}
{/CWContenedorPestanyas}
```

Lista de plugins [254]

B.1.10. CWContenedor

Componente sin funcionalidad visual, se utiliza para albergar tanto tablas de HTML como plugins, desde componentes básicos hasta plugins fichas y/o tablas.

- **Plugins que pueden contener a CWContenedor**
  - Padres
    - CWPanel
- **Plugins que puede contener CWContenedor**
  - Hijos
    - CWTabla
    - CWFichaEdicion
    - CWFicha (sólo en dos casos: Una búsqueda o un campo external en un tabular)

Tabla de argumentos de CWContenedor

El plugin CWContenedor No tiene argumentos

Ejemplos de uso del plugin CWContenedor:

Ejemplo: Contenedor con el componente CWTabla.

```
{CWContenedor}
  {CWTabla ...}
   ...
  {/CWTabla}
{/CWContenedor}
```

Ejemplo: Contenedor con el componente CWFichaEdicion.

```
{CWContenedor}
```

```
  {CWFichaEdicion ...}
   ...
  {/CWFichaEdicion}
{/CWContenedor}
```

Ejemplo: Contenedor para panel de búsqueda.

```
{CWContenedor}
  {CWFicha}
    <br/>Busqueda por fechas<br/>
    {CWCampoTexto nombre="filFechaIni" size="10" editable="true"
 textoAsociado="Fecha recepcion inicial:"}
    {CWCampoTexto nombre="filFechaFin" size="10" editable="true"
 textoAsociado="Fecha recepcion final:"}
  {/CWFicha}
{/CWContenedor}
```

Lista de plugins [254]

B.1.11. CWCheckBox

Equivalente al CHECKBOX de HTML.

- **Plugins que pueden contener a CWCheckBox**
  - Padres
    - CWContenedor
    - CWFila
    - CWFicha
    - CWSolapa
    - CWSelector
- **Plugins que puede contener CWCheckBox**
  - Hijos
    - El plugin CWCheckBox es una hoja (no contiene otros plugins)

Tabla de argumentos de CWCheckBox

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
editable	enumerado	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es edita-

Nombre	Tipo	¿Opcional?	Descripción
			ble y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWCheckBox sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
value	alfanumérico	true	Valor que se le pasa a la capa de negocio cuando el componente este en un panel de búsqueda. Será obligatorio siempre que forme parte de un panel de búsqueda.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá las propiedades del campo: obligatorio, valor chequeado, valor no chequeado. Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.
size	alfanumérico	true	En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
class	string	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.

Ejemplos de uso del plugin CWCheckBox:

Declaración de CWCheckBox como campo de una tabla.

```
{CWCheckBox nombre="descEstado" editable="true" dataType=
$dataType_claseManejadora.descEstado}
```

SUBIR [254]

B.1.12. CWFicha

El plugin ficha, se encarga de dibujar las capas HTML y el campo oculto que indica en que estado se encuentra la ficha (modificación, inserción o borrado). En el caso de los paneles de búsqueda, como en el uso de campos "external" (campos comunes a todas las tuplas de un panel tabular) este plugin tendrá como padre al CWContenedor directamente.

- **Plugins que pueden contener a CWFicha**

- Padres
  - CWFichaEdicion
  - CWContenedor
- **Plugins que puede contener CWFicha**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista
    - CWCheckBox
    - CWSelector
    - CWSolapa

Tabla de argumentos de CWFicha

El plugin CWFicha no tiene argumentos

Ejemplos de uso del plugin CWFicha:

Uso del plugin CWFicha en un panel de edición.

```
{CWFichaEdicion id="FichaEdicion" datos=$smty_datosTablaM}
  {CWFicha}
    {CWCampoTexto nombre="anyo" editable="true" size="4" value=$smty_anyoNuevo
 textoAsociado="A&ntilde;o"}
      
    {CWCampoTexto nombre="nfactura" editable="false" size="6" value="0"
 textoAsociado="N&uacute;mero Factura"}
    <br><br>
    {CWLista nombre="procedencia" radio="true" editable="true" value=
$smty_datosPreInsertadosTinvEntradas2.procedencia textoAsociado="Procedencia"}
  {/CWFicha}
  {CWPaginador enlacesVisibles="3"}
{/CWFichaEdicion}
```

Lista de plugins [254]

B.1.13. CWFichaEdicion

Plugin que alberga dentro un CWFicha, a partir de los datos que le lleguen como parámetros y de su contenido, representa esos datos como fichas, si se desea poder moverse entre ellas, debe incluirse un CWPaginador.

- **Plugins que pueden contener a CWFichaEdicion**
  - Padres

- CWContenedor

- **Plugins que puede contener CWFichaEdicion**
  - Hijos
    - CWFicha
    - CWPaginador

Tabla de argumentos de CWFichaEdicion

Nombre	Tipo	¿Opcional?	Descripción
id	alfanumérico	false	Establace el identificador del CWFichaEdicion
datos	array asociativo	false	Array asociativo con los datos que queremos mostrar en la tabla. Los componentes que representen los datos de este array deben llamarse igual que las claves del array. Será una variable smarty ($smty_datos) que la sustituye IgepPanel.

Ejemplos de uso del plugin CWFichaEdicion:

```
{CWFichaEdicion id="FichaEdicion" datos=$smty_datosFicha}
  {CWFicha}
   ...
  {/CWFicha}
{/CWFichaEdicion
```

Lista de plugins [254]

B.1.14. CWFila

Este plugin representa la fila de una tabla. Se utiliza como molde para manejar la información en modo browse.

- **Plugins que pueden contener a CWFila**
  - Padres
    - CWTabla
- **Plugins que puede contener CWFila**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista
    - CWCheckBox

Tabla de argumentos de CWFila

Nombre	Tipo	¿Opcional?	Descripción
tipoListado	booleano	true	Se utiliza para las tablas que queramos que unicamente muestre la informacion en plan listado, no utilizaremos componentes básicos para representar los datos.

Ejemplos de uso del plugin CWFila:

```
{CWFila tipoListado="false"}
  {CWCampoTexto nombre="lisCif" size="13" textoAsociado="CIF" dataType=
$dataType_Registro.lisCif}
  {CWCampoTexto nombre="lisOrden" size="2" textoAsociado="Orden" dataType=
$dataType_Registro.lisOrden}
  {CWLista nombre="lisDepartamento" textoAsociado="Departamento" value=
$defaultData_Registro.lisDepartamento dataType=$dataType_Registro.lisOrden}
{/CWFila}
```

Lista de plugins [254]

B.1.15. CWImagen

Equivalente al FILE de HTML.

- **Plugins que pueden contener a CWImagen**
  - Padres
    - CWFicha
- **Plugins que puede contener CWImagen**
  - Hijos
    - El plugin CWImagen es una hoja (no contiene otros plugins)

Tabla de argumentos de CWImagen

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
width	entero	true	Indica el ancho de la imagen.
height	entero	true	Indica el alto de la imagen.
alt	enumerado	true	Texto alternativo que se representa cuando la imagen no puede ser mostrada.
src	entero	true	Ruta a la imagen. Cuando se ha de mostrar una imagen de la bd no es obligatorio poner este parámetro.
rutaAbs	booleano	true	Ruta absoluta. Indica que debe buscar la imagen en la ruta absoluta del servidor.
textoAsociado	alfanumérico	false	Texto que acompañará a la imagen.

Nombre	Tipo	¿Opcional?	Descripción
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.
bumpbox	booleano	true	Con "true/false" indicaremos si queremos hacer uso de "Bumpbox". Activándolo, true, al hacer clic en una imagen esta es ampliada en una ventana que toma toda la pantalla con un fondo transparente y en el centro, dentro de un recuadro, que ajusta su tamaño dinámicamente, se muestra la imagen ampliada.

Ejemplos de uso del plugin CWImagen:

```
{CWImagen nombre="fichero" src="imagenes/imagen0.gif" rutaAbs="true"}
```

Lista de plugins [254]

B.1.16. CWLabel

Este plugin tiene diferentes usos, con él podremos implementar:

- (1) Etiqueta <label> de HTML
- (2) Etiqueta <label> de HTML acompañada de un icono css ()
- (3) Etiqueta <a> de HTML acompañada, o no, de un icono css ()
- (4) Etiqueta icono css ()
- **Plugins que pueden contener a CWLabel**
  - Padres
    - CWFicha
    - CWFila

Tabla de argumentos de CWLabel

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá el tipo de dato a mostrar en el campo (cadena, número, fecha...) y sus propiedades como: longi-

Nombre	Tipo	¿Opcional?	Descripción
			tud, obligatorio, máscara, calendario... Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
editable	enumerado	true	Este parámetro solamente tiene sentido cuando estamos utilizando el plugin en el modo (3), es decir, es un enlace. Si su valor es true, el enlace está activo, en cambio, si el valor es false, el enlace no será accesible.
url	alfanumérico	true	Si añadimos este parámetro el plugin generará el modo (3) indicado arriba, un enlace. Podrá contener la url correspondiente al enlace, si esta es fija, (atributo href), o el nombre del campo, si la url viene dada a través de una consulta.
newWindow	string	true	Solamente tiene sentido este parámetro cuando queremos utilizar el plugin como enlace, con él indicaremos que el enlace se abra o no en una ventana nueva.
value	alfanumérico	true	En el caso de que queramos obtener una etiqueta, este parámetro contendrá el texto de la misma, en el caso de un enlace será el texto del enlace. En cambio, en el caso de un icono contendrá la etiqueta css que dibujará el icono.
textoAsociado	alfanumérico	false	Este parámetro solamente es necesario en el caso de que el plugin se utilice en un panel tabular, porque será el que fije el título de la columna.
visible	boolean	true	Con "true/false" indicaremos que queremos forzar si queremos que el campo sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHidra.
class	string	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.
iconCSS	string	false	Contendrá la etiqueta CSS correspondiente al icono que queremos que se muestre.
positionIcon	['left' | 'right']	alfanumerico	En el caso de que haya icono que acompañe a la etiqueta, le podremos indicar la posición en la que aparece respecto a la etiqueta ['left','right'], por defecto el icono aparecerá a la izquierda de la etiqueta.
onlyIcon	string	true	En el caso de que solamente queramos que se dibuje un icono y no añada una etiqueta de texto, este parámetro deberá ponerse a "true".

Ejemplos de uso del plugin CWLabel:

- (1) Etiqueta <label> de HTML

```
      {CWLabel nombre="ediLabel" onlyIcon="false"
 class="classLabel" size="10" editable="true" visible="true" value=
$defaultData_MiClase.ediLabel dataType=$dataType_MiClase.ediLabel}
```

- (2) Etiqueta <label> de HTML acompañada de un icono css ()

```
{CWLabel nombre="ediLabel" onlyIcon="false" iconCSS="glyphicon
glyphicon-eye" positionIcon="right" class="classLabel" size="10"
editable="true" visible="true" value=$defaultData_MiClase.ediLabel
dataType=$dataType_MiClase.ediLabel}
```

- (3) Etiqueta icono css ()

```
{CWLabel nombre="ediLabel" onlyIcon="true" iconCSS="glyphicon
glyphicon-eye" size="10" editable="true" visible="true" value=
$defaultData_MiClase.ediLabel dataType=$dataType_MiClase.ediLabel}
```

- (4) Etiqueta url

```
{CWLabel nombre="ediLabel" onlyIcon="false" url="http://
www.google.es" size="10" editable="true" visible="true" value=
$defaultData_MiClase.ediLabel dataType=$dataType_MiClase.ediLabel}
```

Lista de plugins [254]

B.1.17. CWMaps

Acceso a visores de mapas. Actualmente existen cuatro tipos de visores: OpenStreetMap, IDEE (Infraestructura de Datos espaciales de España), Google Maps e ICVGeo del Instituto Cartográfico Valenciano de la Generalitat Valenciana. El visor OpenStreetMaps es el visor por defecto, y no tiene problemas de licencia ya que el contenido es abierto (para más información http://www.openstreetmap.org/). Respecto al visor IDEE muestra una visualización de cartografía e imágenes de diferentes fuentes de forma interoperable e interactiva. En el caso de Google Maps, recomendamos leer los términos de uso antes de incorporarlo a la aplicación (no es recomendable si la aplicación no está publicada en internet y sin control de acceso). Finalmente, el visor de ICVGeo realiza una conexión a Terrasit, por lo que requiere algo de configuración del sistema.

- **Plugins que pueden contener a CWMaps**
  - Padres
    - CWFicha
- **Plugins que puede contener CWMaps**
  - Hijos
    - El plugin CWMaps es una hoja (no contiene otros plugins)

Tabla de argumentos de CWMaps

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente.
textoAsociado	alfanumérico	false	Texto que acompañará a la imagen.
coordX	alfanumérico	false	Indica el nombre del campo en el que se escribirá la coordenada X (longitud).

Nombre	Tipo	¿Opcional?	Descripción
coordY	alfanumérico	false	Indica el nombre del campo en el que se escribirá la coordenada Y (latitud).
zoom	entero	true	Indica el zoom que se aplicará a la entrada del mapa. Valores más pequeños menor zoom, valores más grandes mayor zoom. Dependerá del visor el rango de valores. Si no se indica por defecto es 10.
visor	alfanumerico	false	Tipo de visor de mapa que se quiere visualizar: "google", "streetmaps", "idee" o "icv".
url	alfanumerico	false	Ruta al visor de mapas (solo para tipo icv).
tipo	alfanumerico	false	Únicamente para el caso del visor Open Street Maps y el visor IDEE. Este parámetro se usa para escoger el tipo de coordenada que se desea utilizar, los dos tipos disponibles para mostrar las coordenadas son UTM que se pasará como parámetro "UTM" y Longitud y Latitud que se pasará como parámetro "lonlat", por defecto se cargará el tipo "lonlat".

Ejemplos de uso del plugin CWMaps:

```
{CWMaps nombre="visor" coordX="edi_LONGITUD" coordY="edi_LATITUD" zoom="8"
 visor="streetmaps" textoAsociado="MAPA" tipo="lonlat"}
```

Lista de plugins [254]

B.1.18. CWInfoContenedor

Plugin que nos permitirá introducir un texto de ayuda y que aparecerá bajo la barra superior,

- **Plugins que pueden contener a CWInfoContenedor**
  - Padres
    - CWContenedor
    - CWFicha
- **Tabla de argumentos de CWInfoContenedor**

  El plugin CWInfoContenedor no tiene parámetros

Ejemplos de uso del plugin CWInfoContenedor:

```
{CWInfoContenedor}

... texto ...

{/CWInfoContenedor}
```

Lista de plugins [254]

B.1.19. CWInformation

Plugin que nos permitirá introducir un texto de ayuda. Introduce una imagen, la que le indiquemos en el parámetro, sobre la que al hacer click aparecerá un bocadillo mostrando la información que le hayamos in-

dicado. A diferencia de la que ofrece CWInfoContenedor que es general, CWInformation ofrece información relativa a cada registro. Esta información puede ser un texto fijo o el valor de un dato de base de datos.

- **Plugins que pueden contener a CWInformation**
  - Padres
    - CWFicha
    - CWFila
- **Tabla de argumentos de CWInformation**

Nombre	Tipo	¿Opcional?	Descripción
imagen	alfanumérico	false	Nombre del fichero que contiene la imagen que queremos que aparezca para indicar que hay información extra, pero sin añadirle la extensión.
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio. El valor de este campo aparecerá en el bocadillo cuando se haga click en la imagen, a no ser que hayamos indicado un texto estático con el parámetro value.
textoAsociado	alfanumérico	true	Texto que acompañará al botón tooltip.
class	alfanumérico	true	Con este parámetro se le podrá asociar un estilo propio.
value	alfanumérico	true	Texto que queremos que aparezca en el bocadillo cuando se haga click en la imagen.

Ejemplos de uso del plugin CWInformation:

```
{CWInformation imagen="58" nombre="cif" value="00000"}
```

Lista de plugins [254]

B.1.20. CWLista

Es un componente de seleccion, simple y/o multiple, se corresponde con los elementos de selección HTML, es decir, las etiquetas SELECT (select-one y select-multiple) y RADIOBUTTON. A través de los argumentos del plugin pueden expresarse selecciones condicionales, por ejemplo, listas dependientes unas de otras (Ej. Provincia - Municipio).

- **Plugins que pueden contener a CWLista**
  - Padres
    - CWFila
    - CWFicha
- **Plugins que puede contener CWLista**
  - Hijos

- El plugin CWLista es una hoja (no contiene otros plugins)

Tabla de argumentos de CWLista

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
editable	enumerado	true	Especifica el comportamiento del selector: si su valor es "true", es editable por el usuario. Si el valor es "false", no es editable y si su valor es "nuevo", será editable solo en la inserción. Si no se especifica el atributo el campo es editable.
datatype	matriz	false	Matriz con una estructura definida en la clase del panel, que definirá las propiedades de la lista, tales como obligatoriedad, tamaño (size), si es una lista múltiple o no, y si queremos que aparezca como un elemento tipo RadioButton o tipo Select. Será una variable smarty en la tpl, definida de la siguiente forma: dataType = $dataType_ClaseManejadora.NombreCampo
numCaracteres	entero	true	Indica el número de caracteres que se mostrarán en la lista desplegable (ancho). En el caso de que nos encontremos en un panel tabular se utilizará este valor como referencia para fijar el ancho de la columna.
autocomplete	integer	true	La lista tendrá la funcionalidad "autocomplete", es decir, a medida que el usuario escribe en ella irán apareciendo aquellos términos que incluyan el texto escrito, el valor introducido será el número mínimo de resultados que debe tener la lista para que funcione la búsqueda a medida que se vayan introduciendo carácteres.
obligatorio	booleano	true	Especifica el comportamiento del componente: si el su valor es 'true', es necesario que el campo tenga valor, o mostrará un mensaje de ALERTA. Si su valor es 'false', no es necesario rellenarlo. cuando el plugin CWLista sea hijo de CWFila o CWFicha. Si no se especifica el atributo, la introducción de valores en el campo no es obligatoria.
multiple	booleano	true	Si se especifica la lista de selección se convierte en una lista de selección multiple.
size	entero	true	Numero máximo de opciones del desplegable sin que aparezca el scroll.
radio	booleano	true	Si aparece este parámetro, el plugin CWLista, genera un grupo de radio Buttoms con las opciones indicadas. No es compatible con el argumento "multiple".
value	matriz	false	Matriz con una estrutura definida, que se le pasa al CWLista para indicar cuales son las opciones posibles, el valor de cada una y cual es la seleccionada. Por lo tanto es una variable smarty ($smty_datosPreinsertados[claseManejadora]) que será sustituida por IgepPanel. Si el componente se encuentra

Nombre	Tipo	¿Opcional?	Descripción
			dentro de un CWTabla o CWFicha, recogerá esos datos del padre (es decir, del CWTabla o del CWFicha). En ese caso éste argumento se utiliza para indicar cuales son los valores que deben mostrarse cuando se inserta un nuevo registro.
actualizaA	alfanumérico	true	Se utiliza este campo en el caso de listas dependientes. Se pondra el nombre del campo que actualizas cuando este toma un valor.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Texto que acompaña a un campo. Si además aparece el argumento obligatorio a true, se le añade un * que indicará que el campo es obligatorio de rellenar.
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el principio. En lugar de obedecer el comportamiento prefijado por gvHigra.
class	string	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.

Ejemplos de uso del plugin CWLista:

Ejemplo: Dos listas dependientes.

```
{CWLista nombre="ediCodProv" radio="true" size="3" actualizaA="ediCodMun"
 editable="true" value=$defaultData_Registro.ediCodProv dataType=
$dataType_Registro.ediCodProv}
{CWLista nombre="ediCodMun" size="3" editable="true" dataType=
$dataType_Registro.ediCodMun value=$defaultData_Registro.ediCodMun}
```

Lista de plugins [254]

B.1.21. CWMarcoPanel

Se utiliza para agrupar paneles, cada panel representa un modo de trabajo (busqueda, ficha, browse...) sobre un mismo conjunto de datos o un subconjunto de los mismos. Sus hijos serán siempre uno o más paneles y por último, un CWContenedorPestanyas si es que estas deben aparecer.

- **Plugins que pueden contener a CWMarcoPanel**
  - Padres
    - CWVentana
- **Plugins que puede contener CWMarcoPanel**
  - Hijos
    - CWPanel

- CWContenedorPestanyas

Tabla de argumentos de CWMarcoPanel

Nombre	Tipo	¿Opcional?	Descripción
conPestanyas	booleano	true	Debe utilizarse en cuanto aparezcan los modos de comportamiento. Cuando su valor es true, introduce los elementos Javascript necesarios para el manejo de las pestañas.

Ejemplos de uso del plugin campo:

Ejemplo: Uso del componente CWMarcoPanel en un Maestro-Detalle.

```
{CWVentana ...}
...
 {CWMarcoPanel conPestanyas="true"}

<!--*********** PANEL fil *****************-->
   {CWPanel id="fil" action="buscar" method="post" estado="$estado_fil"
 claseManejadora="Registro"}
   ...
   {/CWPanel}
<!--*********** PANEL lis *****************-->
   {CWPanel id="edi" tipoComprobacion="envio" action="operarBD" method="post"
 estado="$estado_edi" claseManejadora="Registro"}
   ...
   {/CWPanel}
<!-- ***************** PESTANYAS ***********************-->
   {CWContenedorPestanyas}
     {CWPestanya tipo="fil" estado=$estado_fil}
     {CWPestanya tipo="edi" estado=$estado_edi}
   {/CWContenedorPestanyas}
 {/CWMarcoPanel}
{/CWVentana}
```

Lista de plugins [254]

B.1.22. CWMenuLayer

Adaptación simplificada del menu dinámico generado en PHP y JavaScript del proyecto GPL PHPLM (PHP Layers Menu) de Marco Pratesi

- **Plugins que pueden contener a CWMenuLayer**
  - Padres
    - CWBarra
- **Plugins que puede contener CWMenuLayer**
  - Hijos
    - El plugin CWMenuLayer es una hoja (no contiene otros plugins)

Tabla de argumentos de CWMenuLayer

Nombre	Tipo	¿Opcional?	Descripción
name	alfanumérico	false	Especifica el nombre del componente. Si no se indica, se genera uno automáticamente. Es una variable smarty ($smty_nombre) que se sustituirá en IgepPanel.
usarImagenesAplicacion	booleano	false	Parámetro opcional, cuyo valor por defecto es "false", indica si debe buscar las imágenes del menu en la aplicación, o en caso de ser falso, utilizará las imágenes de gvHidra.
imgDescenso	alfanumérico	true	Indica el nombre de la imagen (soporta gif, jpg o png) que despliega el menu de forma descedente. La imagen por defecto es down-arrow.png
imgDespliega	alfanumérico	true	Indica el nombre de la imagen (soporta gif, jpg o png) que despliega el menu de forma lateral. La imagen por defecto es forward-arrow.png
fichero	alfanumérico	true	Parámetro opcional con el que se indica el fichero (extensión .str) que contiene la estructura del menu el formato del mismo es: ". | opcion | url | texto ayuda emergente | imagen" Si se utiliza esta forma de establecer el menú no ha de existir el próximo parámetro 'cadenaMenu'.
cadenaMenu	alfanumérico	true	Cadena de texto con la estructura del menú. Al ser una cadena de texto, podemos generar menús dinámicos (será un fichero xml), es decir, cuyas opciones varíen en función de algún parámetro, por ejemplo en función del ROL del usuario etc... Será una variable smarty $smty_cadenaMenu que se sustituirá en IgepPantalla.

Ejemplos de uso del plugin CWMenuLayer:

Ejemplo de la estuctura del menú

```
.|Menu|
..|Entrada Opción 1||
...|Inserción|?view=views/insercion.php| Opción para Nuevo Registro |
 insertar.png
...|Mantenimiento|?view=views/mantenimiento.php| Opción para Mantenimiento
..|Entrada Opción 2||
...|Prueba con frame oculto|?view=views/inicio.php| Aplicación
...|Prueba Maestro-Detalle|?view=views/MD.php| Home en flaco
```

Ejemplo: Dentro de CWVentana.

```
{CWVentana tipoAviso=$smty_tipoAviso codAviso=$smty_codError descBreve=
$smty_descBreve textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
  {CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=
$smty_customTitle}
    {CWMenuLayer name="$smty_nombre" cadenaMenu="$smty_cadenaMenu"}
  {/CWBarra}
  {CWMarcoPanel conPestanyas="true"}
   ...
  {/CWMarcoPanel}
{/CWVentana}
```

Lista de plugins [254]

B.1.23. CWPaginador

Integra una línea de enlaces para paginar cuando se presentan múltiples regitros a través de los plugins CWTabla y CWFicha.

- **Plugins que pueden contener a CWPaginador**
  - Padres
    - CWTabla
    - CWFichaEdicion
- **Plugins que puede contener CWPaginador**
  - Hijos
    - El plugin CWPaginador es una hoja (no contiene otros plugins)

Tabla de argumentos de CWPaginador

Nombre	Tipo	¿Opcional?	Descripción
pagInicial	entero	true	Indica la página que aparecerá visible cuando se cargue la pantalla, por defecto siempre vale 0.
enlacesVisibles	entero	true	Indica el número de enlaces que apareceran para realizar la navegación además de los fijos (siguiente, ultimo, anterior y primero)
iconCSS	boolean	true	Indicamos que los iconos del paginador son icon-fonts

Ejemplos de uso del plugin CWPaginador:

Ejemplo: Utilizacion de un CWPaginador en una tabla.

```
{CWTabla conCheck="true" seleccionUnica="true" id="Tabla1" datos=
$smty_datosTabla}
  {CWFila tipoListado="false"}
    {CWCampoTexto nombre="lisCif" size="9" editable="true" textoAsociado="CIF"
 dataType=$smty_dataType_Personas.lisCif}
    {CWCampoTexto nombre="lisNombre" editable="true" size="30"
 textoAsociado="Nombre" dataType=$smty_dataType_Personas.lisNombre}
  {/CWFila}
  {CWPaginador enlacesVisibles="3"}
{/CWTabla}
```

Lista de plugins [254]

B.1.24. CWPanel

Este componente es un contenedor, aporta javascript, una tabla HTML para incluir otros plugins, y un elemento FORM de HTML, por lo que es el plugin que realiza los envios de información hacia la lógica. Cada panel es un formulario HTML y se corresponde con una de las pestañas de los modos de trabajo indicados en la guía de estilo (búsqueda, ficha, tabular).

El orden de los hijos en un panel si que es importante y debe respetarse.

- **Plugins que pueden contener a CWPanel**
  - Padres
    - CWMarcoPanel
- **Plugins que puede contener CWPanel**
  - Hijos (hay que respetar este orden a la hora de crear los hijos)
    - CWBarraSupPanel
    - CWContenedor
    - CWBarraInfPanel

Tabla de argumentos de CWPanel

Nombre	Tipo	¿Opcional?	Descripción	Valores
id	enumerado	false	Fija el identificador del componete y del formulario HTML que incluye por debajo. Es necesario que dicho nombre sea único para evitar comportamientos no previstos en la interfaz.	Puede tomar los siguientes valores: • *fil*: panel de búsqu da. • *edi*: panel registro • *lis*: panel tabulare • *ediMaestro / ediD talle*: panel regist maestro o detalle, respectivamente. • *lisMaestro / lisDe talle*: panel tabula maestro o detalle, respectivamente.
action	alfanumérico	true	Tipo de acción que debe coincidir con la correspondiente en el mappings.	
accion	alfanumérico	true	Estado en el que se quiere encontrar el panel al cargarlo (modo inserción, modificación, borrado)	Puede tomar los siguientes valores: • *insertar* • *modificar* • *borrar*
tipoComprobacion	alfanumérico	false	Indica la comprobación que queremos que se realice a los campos que tengan el argumento	Puede tomar los siguientes valores: • *envio*

Nombre	Tipo	¿Opcional?	Descripción	Valores
			comprobacion igual true. Por defecto hará una comprobación "envio"	• *foco* • *todo*
method	alfanumérico	false	Indica la manera en que debe realizarse el submit del FORM HTML (get/post).	
estado	alfanumérico	true	Indica el estado del panel (activo o desactivado) cuando se carga la pantalla. Las opciones son, si no viene dado desde la capa de negocio con variables smarty:	Puede tomar los siguientes valores: • *on*: visible y activ • *off*: visible e inact • *inactivo*: no visib inactivo
claseManejadora	alfanumérico	true	Indica la clase que va a ocuparse de la lógica de esta pantalla.	

Ejemplos de uso del plugin CWPanel:

Ejemplo de uso del action:

```
-> Parámetro del plugin
{CWPanel id="edi" action="operarBD" estado="$estado_edi"
 claseManejadora="TinvEntradas2" ...}

-> Correspondencia en el mappings.php
$this->_AddMapping('TinvEntradas2__operarBD', 'TinvEntradas2', '', 'IgepForm',
 0);
```

Ejemplo: Uso del componente CWPanel en un panel de búsqueda.

```
{CWPanel id="fil" action="buscar" method="post" estado="$estado_fil"
 claseManejadora="TinvDonantes"}
  {CWBarraSupPanel titulo="Donantes"}
    {CWBotonTooltip imagen="01" titulo="Insertar registro" funcion="insertar"
 actuaSobre="ficha" action="nuevo"}
    {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar"
 actuaSobre="ficha"}
  {/CWBarraSupPanel}
  {CWContenedor}
    {CWFicha}
      <br>  
      {CWCampoTexto nombre="filCif" editable="true" size="13"
 textoAsociado="CIF/NIF" dataType=$dataType_TinvDonantes.filCif}
      <br><br>
    {/CWFicha}
   {/CWContenedor}
   {CWBarraInfPanel}
     {CWBoton imagen="50" texto="Buscar" class="boton" accion="buscar"}
   {/CWBarraInfPanel}
{/CWPanel}
```

Lista de plugins [254]

B.1.25. CWPantallaEntrada

Con este plugin creamos la pantalla inicial de cualquier aplicación según la guía de estilo. Todos sus parámetros son variables smarty que se sustituyen internamente.

- **Plugins que pueden contener a CWPantallaEntrada**
  - Padres
    - CWVentana
- **Plugins que puede contener CWPantallaEntrada**
  - Hijos
    - El plugin CWPantallaEntrada es una hoja (no contiene otros plugins)

Tabla de argumentos de CWPantallaEntrada

Nombre	Tipo	¿Opcional?	Descripción
usuario	alfanumérico	false	Se utiliza para mostrar en pantalla el usuario que está conectado a la aplicación.
nomApl	alfanumérico	false	Se utiliza para mostrar el nombre de la aplicación que se va a utilizar.
codApl	alfanumérico	false	Se utiliza para mostrar en pantalla el código de la aplicación. (Abreviatura que suele corresponderse con el nombre del directorio del servidor Web donde se ubica)

Ejemplos de uso del plugin CWPantallaEntrada:

Ejemplo: Declaración de una Pantalla de inicio. Sólo aparece en aplicación.tpl, plantilla exlclusiva de igep.

```
{CWVentana tipoAviso=$smty_tipoAviso titulo=$smty_tituloApl codAviso=
$smty_codError descBreve=$smty_descBreve textoAviso=$smty_textoAviso onLoad=
$smty_jsOnLoad onUnload=$smty_jsOnUnload}
  {CWPantallaEntrada usuario=$smty_usuario rolApl=$smty_rolApl nomApl=
$smty_aplicacion codApl=$smty_codaplic}
{/CWVentana}
```

Lista de plugins [254]

B.1.26. CWPestanyas

Plugin que dibujará la pestaña lateral correspondiente.

- **Plugins que pueden contener a CWPestanyas**
  - Padres
    - CWContenedorPestanyas
- **Plugins que puede contener CWPestanyas**

- Hijos
  - El plugin CWPestanyas es una hoja (no contiene otros plugins)

Tabla de argumentos de CWPestanyas

Nombre	Tipo	¿Opcional?	Descripción	Valores
tipo	enumerado	false	Indicará el tipo de pestaña	El valor a tomar depende nel al que va asociado: • *fil*: Panel filtro • *edi*: Panel registro • *lis*: Panel tabular
estado	alfanumérico	false	Valor que corresponderá con el estado que queramos que aparezaca en un principio. Puede ser establecida por defecto en la tpl o ser una variable de smarty sustituida internamente.	
panelAsociado	alfanumérico	false	Nombre del panel al que está asociada dicha pestaña.	
ocultar	alfanumérico	false	Se utiliza en los mantenimientos mestro detalle, nos permite ocultar el panel de detalle cuano se se selecciona la pestaña en el que está definido el parámetro (el valor es "Detalle")	
mostrar	alfanumérico	false	Similar a ocultar, pero en este caso se utiliza para activar el detalle ('Detalle' como argumento) al seleccionar la pestaña que tiene dicho parámetro	

Ejemplos de uso del plugin CWPestanyas:

Ejemplo: Definición de dos pestañas.

```
{CWContenedorPestanyas}
  {CWPestanya tipo="fil" estado=$estado_fil}
  {CWPestanya tipo="lis" estado=$estado_lis}
{/CWContenedorPestanyas}
```

Lista de plugins [254]

B.1.27. CWRichAreaTexto

Equivalente al TEXTAREA de HTML pero enriquecido. Se desaconseja el uso de este plugin en paneles de búsqueda y en tabulares. Se ha de tener la precaución de que al formatear el texto, el tamaño puede ocupar más de lo reservado en base de datos para ese campo.

- **Plugins que pueden contener a CWRichAreaTexto**
  - Padres
    - CWContenedor

- CWFicha
- CWSelector
- CWSolapa

- **Plugins que puede contener CWRichAreaTexto**
  - Hijos
    - El plugin CWRichAreaTexto es una hoja (no contiene otros plugins)

Tabla de argumentos de CWRichAreaTexto

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
css	alfanumérico	true	Por defecto tendrá el siguiente tamaño: width: 600px; height: 175px;. Si se quiere modificar, será necesario crear un estilo para el CWRichTextArea en appStyle.css, y el nombre del estilo será el que se pase en este parámetro.
value	alfanumérico	true	Valor por defecto que aparecera en el campo.
editable	alfanumérico	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWAreaTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Fija el texto descriptivo que acompaña a un campo (etiqueta). Si además aparece el argumento/parámetro "obligatorio" a true, se le añade un * que indicará que el campo es obligatorio de rellenar. El texto que se incorpora como etiqueta, será también el que se utilice en los mensajes de comprobación de cmapos obligatorios, etc...
mostrarTextoAsociado	booleano	false	No siempre queremos que se muestre la etiqueta o texto asociado a un campo, aunque puede interesarnos que dicho texto exista en realidad para utilizarlo en mensajes con el usuario. La solución en utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
actualizaA	alfanumérico	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
visible	booleano	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisible desde el

Nombre	Tipo	¿Opcional?	Descripción
			principio. En lugar de obedecer el comportamiento prefijado por gvHigra.
class	string	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.

Ejemplos de uso del plugin CWRichAreaTexto:

Ejemplo: Declaración de CWRichAreaTexto.

```
{CWRichAreaTexto nombre="comentario" textoAsociado="Expedientes"
 css="styleComentario"}
```

Lista de plugins [254]

B.1.28. CWSelector

Define un patrón tipo lista que nos permita simular el maestro-detalle-subdetalle. Por un lado tenemos una lista múltiple donde se irán acumulando los valores procedentes de los campos que se hayan incluido en el selector. Estos campos pueden ser un conjunto de elementos básicos (CWCampoTexto, CWLista (NO multiple), CWCheckBox, CWBotonToolTip) o una única lista multiple. No se podrá combinar una lista múltiple con otros elementos de formulario (campos de texto, listas simples...), ya que las opciones elegidas de la lista múltiple se copiarán a la lista destino como opciones diferentes y no una combinación como la que se hace con varios campos.

- **Plugins que pueden contener a CWSelector**
  - Padres
    - CWFicha
- **Plugins que puede contener CWSelector**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista
    - CWCheckBox
    - CWBotonToolTip

Tabla de argumentos de CWSelector

Nombre	Tipo	¿Opcional?	Descripción
titulo	alfanumérico	false	Título que identifica el elemento.
botones	matriz	false	Estructura recibida de la capa de negocio para indicar que botones ('insertar','modificar','eliminar') aparecerán visibles.
nombre	alfanumérico	false	Nombre para el campo destino donde se irán acumulando los valores.

Nombre	Tipo	¿Opcional?	Descripción
editable	alfanumérico	true	Especifica el comportamiento del campo destino: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWAreaTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
separador	alfanumérico	true	Carácter que utilizaremos para separar los distintos datos que formen parte de un mismo registro, por defecto será el carácter "|".
value	matriz	true	Matriz con una estrutura definida, que se le pasa al CWSelector para mostrar los datos existentes a los que podremos añadir o eliminar otros. Será una variable smarty $smty_datos que se encargará IgepPanel de sustituir.
rows	entero	true	Especifica el número de filas que tendrá el textarea.

Ejemplos de uso del plugin CWSelector:

Ejemplo: Selector con varios campos de texto.

```
{CWSelector titulo="*Formato:" botones=$smty_botones nombre="listaBienes"
 editable="nuevo" separador=","}
    {CWCampoTexto nombre="ediCcentro" editable="nuevo" size="2"
 textoAsociado="Centro" actualizaA="ediDcentro" dataType=
$dataType_MiClase.ediCcentro}
      
    {CWCampoTexto nombre="ediDcentro" editable="false" size="50" dataType=
$dataType_MiClase.ediDcentro}
    {CWBotonTooltip imagen="13" titulo="Ventana de seleccion"
 funcion="abrirVS" actuaSobre="ccentro"}
    <br><br>
    {CWCampoTexto nombre="ediUds" editable="true" size="5" textoAsociado="Uds"
 dataType=$dataType_MiClase.ediUds}
    <br><br>
{/CWSelector}
```

Ejemplo: Selector con una lista con datos de la BD.

```
{CWSelector titulo="*Formato:" botones=$smty_botones nombre="listaBienes"
 editable="nuevo" value=$defaultData_Registro.listaBienes}
    {CWLista multiple=true nombre="categorias" editable="nuevo"
 textoAsociado="Categorias" value=$defaultData_Registro.categorias}
{/CWSelector}
```

Lista de plugins [254]

B.1.29. CWSlider

Plugin que nos aportará el poder incorporar un visor de imágenes en un panel registro. En esta primera versión, el plugin obtiene las imágenes de un directorio y no de BD.

- **Plugins que pueden contener a CWSlider**

- Padres
  - CWFicha

Tabla de argumentos de CWSlider

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Fija el identificador del componente. Es necesario que dicho nombre sea único para evitar comportamientos no previstos en la interfaz.
images	array	false	Array de imágenes que aparecerán en el visor slider.

Ejemplos de uso del plugin CWSlider:

Ejemplo: Declaración de un visor slider en una ficha.

```
{CWPanel id="edi" action="operarBD" estado="$estado_edi"
claseManejadora="RegistroSlider"}
{CWContenedor}
       {CWFichaEdicion id="FichaEdicion" datos=$smty_datosFicha}
    {CWFicha}
   {CWSlider  nombre="myImages" images=$defaultData_RegistroSlider.myImages}
  {/CWFicha}
 {/CWFichaEdicion}
{/CWContenedor}
{/CWPanel}
```

Lista de plugins [254]

B.1.30. CWSolapa

Plugin que alberga dentro las solapas asociadas a un panel. Genera la lógica JavaScript necesaria para manejarlas. Es necesario que las solapas estén dentro de una FichaEdicion, que tiene como parámetros 'numSolapas' y 'titulosSolapas' (ver plugin CWFichaEdicion).

- **Plugins que pueden contener a CWSolapa**
  - Padres
    - CWFicha
- **Plugins que puede contener CWSolapa**
  - Hijos
    - CWCampoTexto
    - CWAreaTexto
    - CWLista

- CWCheckBox

Tabla de argumentos de CWSolapa

Nombre	Tipo	¿Opcional?	Descripción
titulo	alfanumérico	false	Establace el texto que aparecerá en la solapa.
poscionSolapa	entero	false	Es un atributo obligatorio, que indica la posicion que tendra esa solapa. Debe ser consecutivo y empezar por 0.

Ejemplos de uso del plugin CWSolapa:

Ejemplo: Declaración de un CWContenedorPestanya con dos pestanyas dentro.

```
{CWFichaEdicion id="FichaEdicion" datos=$smty_datosFicha}
  {CWFicha}
    {CWSolapa titulo=" Datos 1" posicionSolapa=0}
      {CWCampoTexto nombre="ediCif" editable="true" size="13"
 textoAsociado="CIF" dataType=$dataType_Registro.ediCif}
      {CWCampoTexto nombre="ediOrden" editable="true" size="2"
 textoAsociado="Orden" dataType=$dataType_Registro.ediOrden}
    {/CWSolapa}
    {CWSolapa titulo="Datos personales" posicionSolapa="1"}
       ...
    {/CWSolapa}
{/CWFicha}
{/CWFichaEdicion}
```

Lista de plugins [254]

B.1.31. CWTabla

Equivalente al TABLE del HTML

- **Plugins que pueden contener a CWTabla**
  - Padres
    - CWContenedor
- **Plugins que puede contener CWTabla**
  - Hijos
    - CWFila

Tabla de argumentos de CWTabla

Nombre	Tipo	¿Opcional?	Descripción
id	alfanumérico	false	Atributo utilizado por plugins herederos o descendientes. Lo utiliza el plugin CWFila para generar los elementos TR con un id.

Nombre	Tipo	¿Opcional?	Descripción
datos	array asociativo	false	Vector asociativo que el programador pasa a la tabla con los datos que queremos que se muestre en esta. Variable smarty ($smty_datos) que se sustituye internamente.
conCheck	booleano	true	Si aparece y su valor es true, al principio de cada fila que compone la tabla aparecera un checkbox que utilizaremos para seleccionar la fila o filas sobre las que realizaremos las diferentes acciones.
seleccionUnica	booleano	true	Permite un único elemento seleccionado en la tabla.
conCheckTodos	booleano	true	Si aparece y su valor es true, aparecera un checkbox en la cabecera de la tabla que nos permitira seleccionar y deseleccionar todos los registros.
colResizable	booleano	false	Atributo utilizado por plugins descendientes, en este caso lo utiliza el plugin CWFila para las columnas con posibilidad de redimensión.
numFilasPantalla	entero	true	Fija el número de filas de datos que queremos que aparezca por pantalla. En caso de no aparecer por defecto apareceran 6 filas de datos.

Ejemplos de uso del plugin CWTabla:

```
{CWTabla conCheck="true" seleccionUnica="true" id="Tabla1" datos=
$smty_datosTabla}
  {CWFila tipoListado="false"}
     {CWCampoTexto nombre="lisCif" size="9" editable="true"
 textoAsociado="CIF" dataType=$dataType_MiClase.lisCif}
     {CWCampoTexto nombre="lisNombre" editable="true" size="30"
 textoAsociado="Nombre" dataType=$dataType_MiClase.lisNombre}
     {CWCampoTexto nombre="lisMoto" editable="true" size="30"
 textoAsociado="Moto" dataType=$dataType_MiClase.lisMoto}
  {/CWFila}
  {CWPaginador enlacesVisibles="3"}
{/CWTabla}
```

Lista de plugins [254]

B.1.32. CWTree

Define un botón tooltip que abrirá una estructura jerárquica en forma de árbol en una ventana modal. De ese árbol se podrá seleccionar una o más opciones que serán trasladadas al componente que se haya indicado de la ventana padre. El valor seleccionado del árbol se compone de un identificador y una descripción, por lo tanto, si este valor es copiado en una lista no hay que hacer nada extra ya que el formato de la lista puede guardar ambos valores (en el value se guardará el identificador, y la descripción será el descriptivo de la lista). En cambio si se opta porque el valor se copie en un CWCampoTexto, la opción de almacenar el identificador se ha optado porque se cree en la tpl un campo oculto con el mismo nombre del CWCampoTexto pero con el prefijo "tree_". La estructura del árbol se debe generar en formato JSON en una acción particular, y debe cumplir el siguiente patrón: [{ "text": "Edificio Principal", "id": 1, "tags": [10], "state": { "checked": true, "disabled": false, "expanded": false, "selected": false }, "nodes": [{ "text": "Planta 1a", "id": 11, "nodes": [{ "text": "Sala 1a", "id": 111 }, { "text": "Sala 2a", "id": 112 }] }, { "text": "Planta 2a", "id": 12 }] }, { "text": "Edificio secundario", "id": 2, "tags": [5], "state": { "checked": false, "disabled": false, "expanded": false, "selected": false }, "nodes": [{ "text": "Planta 1a", "id": 21 }] }, { "text": "Edificio

adjunto", "id": 3, "tags": [1], "state": { "checked": false, "disabled": false, "expanded": false, "selected": false } }, { "text": "Edificio adjunto 2", "id": 4, "tags": [10], "state": { "checked": false, "disabled": false, "expanded": false, "selected": false } }]

- **Plugins que pueden contener a CWTree**
  - Padres
    - CWFicha
    - CWFila

Tabla de argumentos de CWTree

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Atributo utilizado para identificar el botón tooltip que abrirá la ventana así como el árbol.
multiple	booleano	false	Indicará si es posible la selección múltiple o no.
editable	booleano	true	Indicará el estado del botón tooltip que abre la ventana modal con el árbol.
visible	booleano	true	Indicará si el botón tooltip es visible o no.
actualizaA	alfanumérico	true	Nombre del campo donde se copiará el valor seleccionado en el árbol.
action	alfanumérico	false	Acción particular que se ejecutará al abrirse la ventana modal del árbol. En la acción particular se creará la estructura JSON del árbol.

Ejemplos de uso del plugin CWTree:

```
{CWTabla conCheck="true" seleccionUnica="true" id="Tabla1" datos=
$smty_datosTabla}
  {CWFicha}
    {CWCampoTexto nombre="campoAjax" dataType=$dataType_TreeSelector.campoAjax
 value=$defaultData_TreeSelector.campoAjax textoAsociado="Valor seleccionado"}
    {CWCampoTexto nombre="tree_campoAjax" oculto="true" dataType=
$dataType_TreeSelector.campoAjax value=$defaultData_TreeSelector.campoAjax}
 {CWTree textoAsociado="Árbol" multiple="false" nombre="arbolAjax"
 editable="true" visible="true" dataType=$dataType_TreeSelector.arbolAjax
 actualizaA="campoAjax" action="tree"}
  {/CWFicha}
```

Lista de plugins [254]

B.1.33. CWUpLoad

Equivale al FILE de HTML

- **Plugins que pueden contener a CWUpLoad**
  - Padres
    - El plugin CWUpLoad es raiz (no lo contiene ningún otro plugin)

- **Plugins que puede contener CWUpLoad**
  - Hijos
    - CWFicha
    - CWConenedor

Tabla de argumentos de CWUpLoad

Nombre	Tipo	¿Opcional?	Descripción
nombre	alfanumérico	false	Nombre para identificar la instancia del componente. Si los datos que maneja son persistentes (acceso a BD), es necesario que este parámetro coincida con el definido por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
obligatorio	booleano	true	Especifica el comportamiento del campo: Si el su valor es true, es necesario que el campo tenga valor, o mostrará un mensaje de ALERTA. Si su valor es false, no es necesario rellenarlo. Cuando el plugin CWCampoTexto sea hijo de CWFila o CWFicha, si no se especifica el atributo, la introducción de valores en el campo no es obligatoria.
editable	booleano	true	Especifica el comportamiento del campo: si su valor es true, es editable por el usuario. Si el valor es false, no es editable y si su valor es nuevo, será editable solo en la inserción cuando el plugin CWCampoTexto sea hijo de CWFila o CWFicha. Si no se especifica el atributo, el campo es editable.
visible	enumerado	true	Especifica si el componente es visible o no.
tabIndex	entero	true	Especifica el orden de tabulación.
textoAsociado	alfanumérico	false	Texto que acompaña a un campo. Si además aparece el argumento obligatorio a true, se le añade un * que indicará que el campo es obligatorio de rellenar.

Ejemplos de uso del plugin CWUpLoad:

```
{CWUpLoad nombre="ficheroUpload" size="10" textoAsociado="Si quieres cambiar
 la imagen..."}
```

Lista de plugins [254]

B.1.34. CWVentana

Este componente se basa en la ventana HTML. Es el plugin raíz, con el se incluye la base javascript necesaria para el comportamiento de la interfaz (manejo de capas, errores...). Todos sus parámetros serán fijados internamente mediante variables smarty.

- **Plugins que pueden contener a CWVentana**
  - Padres
    - El plugin CWVentana es raiz (no lo contiene ningún otro plugin)

- **Plugins que puede contener CWVentana**
  - Hijos
    - CWBarra
    - CWMarcoPanel
    - CWPantallaEntrada
    - CWEjecutarScripts

Tabla de argumentos de CWVentana

Nombre	Tipo	¿Opcional?	Descripción	Valores
titulo	alfanumérico	true	Fija el título de la Ventana HTML. Variable smarty $smty_tituloApl.	
tipoAviso	alfanumérico	false	Indica el tipo de aviso según la guia de estilo. Variable smarty $smty_tipoAviso.	Los tipos de aviso pueden ser: • *aviso* • *alerta* • *notificación* • *sugerencia*
codAviso	alfanumérico	false	Fija el código de aviso. Variable smarty $smty_codError.	
descBreve	alfanumérico	false	Descripción breve del mensaje del aviso. Variable smarty $smty_descBreve.	
textoAviso	alfanumérico	false	Descripción detallada del aviso. Variable smarty $smty_textoAviso.	
onload	alfanumérico	false	En este parámetro podremos introducir una llamada a una funcion javascript que debe ejecutarse en el evento onLoad de la página	
onUnload	alfanumérico	false	En este parámetro podremos introducir una llamada a una funcion javascript que debe ejecutarse en el evento onUnLoad de la página	
manageSession	boolean	false	Con este parámetro indicaremos si se quiere mantener la sesión activa o no para la ventana.	

Ejemplos de uso del plugin CWVentana:

```
{CWVentana tipoAviso=$smty_tipoAviso  codAviso=$smty_codError  descBreve =
 $smty_descBreve  textoAviso=$smty_textoAviso onLoad=$smty_jsOnLoad}
...
{/CWVentana}
```

Lista de plugins [254]

Apéndice C. Listados Jasper en gvHIDRA

C.1. Integracion de Jasper en un mantenimiento

La manera de integrar un informe Jasper dentro de una aplicación gvHIDRA es similar a la que se utiliza para integrar otros informes (listadosOO, informesOD, listadosXML), o incluso más sencilla, puesto que la fuente de datos del informe se gestiona de forma interna (la consulta sql está embebida en el informe).

Supongamos que trabajamos en un proyecto que necesita nuestro proyecto se llama “pruebas-igep”, detallemos poco a poco los pasos a seguir:

1. Importar el proyecto Jasper. Para ello, el directorio “include” de nuestro proyecto, tendrá en su interior un directorio “jasper”. En dicho directorio, podremos encontrar tres subdirectorios básicos:

   - Directorio “ejemplos”: Distintos ejemplos disponibles.
   - Directorio “jars”: Clases Java que realizan la invocación del informe y la comunicación con las clases PHP.
   - Directorio “modulosPHP”: Clases PHP que se ofrecen al usuario (programador), encargadas de preparar la invocación.

   Es importante que incluiyamos en el fichero .cvsignore del proyecto una línea con la palabar "jasper" para evitar que el proyecto jasper se incorpore a la versión del CVS de su proyecto.

2. Crear en el raíz de nuestro proyecto el directorio “plantillasJasper”, donde dejaremos los ficheros “.jasper” que vayamos compilando con el iReport (los informes compilados). Durante el desarrollo del proyecto, podemos colocar los ficheros "fuente" de los informes (extensión “.jrxml”) bien, en el mismo directorio o bien en un subdirectorio llamado "fuentes". Dichos ficheros son ficheros XML editables con el iReport. Es MUY IMPORTANTE no incluir dichos ficheros (los de extensión “.jrxml”) en la versión de produccion de la aplicación, dado que aparecen datos de conexión en su interior (usuario y contraseña de acceso a BD) en texto plano. Por lo tanto, cuando el responsable de una aplicación realice el despliegue o instalación de la misma en producción, debería eliminar los ficheros .jrxml” del directorio "plantillasJasper". La figura siguiente muestra la jerarquía de directorios de la que hemos hablado:

3. Copiar el fichero compilado (fichero de extensión “.jasper”) del informe dentro del directorio “plantillasJasper”.

4. Crear un mantenimiento de gvHIDRA siguiendo el patrón P1M1(FIL). Una vez creada la ventana, tenemos que incorporar en ella una accion particular para que el listado se lance. En nuestro ejemplo abrirListado.

```
1:<?php
2:
3:require_once('include/jasper/modulosPHP/informeJasper.php');
···SIGUE···
7:class claseDemo extends gvHidraForm
8:{
```

```
9:
10:  var $infJasper_demo;
···SIGUE···
27: function accionesParticulares($str_accion, $objDatos)
28: {
29:    $objDatos->setOperation('external');
30:    $informeJ = new InformeJasper('DemoFacturas');
31:    $informeJ->setDataSourceType('sgbd');
32:    $conf = ConfigFramework::getConfig();
33:  $g_dsn = $conf->getDSN('g_dsn');
34:    $informeJ->importPearDSN($g_dsn);
35:    //Si la conexión es distinta podemos especificarlo, por
 ejemplo:
36:    /*
37:     $informeJ->setDBType('pgsql'); $informeJ-
>setDBHost('localhost');
38:     $informeJ->setDBPort('5432'); $informeJ-
>setDBDatabase('marte');
39:     $informeJ->setDBUser('prueba'); $informeJ-
>setDBPassword('foo');
40:    */
41:    $informeJ->setJasperFile('./plantillasJasper/
DemoFacturas.jasper');
42:    //Comprobamos los parámetros del informe y los pasamos...
43:    if ($objDatos->getValue('anyo')!='')
44:    {
45:        $informeJ->addParam('Anyo', $objDatos->getValue('anyo'),
 'String');
46:    }
··· Comprobación del resto de parámetros ···
61:    $this->infJasper_demo = & $informeJ;
62:    $actionForward = $objDatos->getForward('mostrarListado');
63:    $this->openWindow($actionForward);
64:    $this->showMessage('APL-10');
65:    $actionForward = $objDatos->getForward('correcto');
66:    return $actionForward;
67: }//Fin accionesParticulares
···SIGUE···
123:}//Fin
124:?>
```

Del código anterior debemos destacar:

- Línea 10: Declaramos un atributo de la clase que será una referencia al informe Jasper. Se crea como atributo de la clase, porque es la manera más sencilla de hacerlo accesible a la vista.

- Líneas 30-34: Declaramos un nuevo informe, de tipo SGBD (acceso a BD relacional) e importamos los parámetros de conexión que necesita desde la conexión PEAR activa. Si el informe se conectase a una BD distinta, podríamos especificarla también usando las funciones que hay comentadas en las líneas 35-39.

- Línea 41: Declaramos el nombre del fichero compilado (fichero de extensión “.jasper”) que hemos colocado dentro del directorio “plantillasJasper” que queremos que se abra.

- Lineas 43-46: Si el informe jasper tiene parámetros de entrada, en esta línea vemos cómo podemos dar valor a uno de los parámetros que espera el informe jasper. Los parámetros de la función son:
  - Nombre del parámetro: deberá coincidir con el nombre que hemos dado de alta en iReports
  - Valor del parámetro: enviaremos el valor (ojo con tipos complejos, como las fechas, eque en PHP son objetos).
  - Valor del parámetro: enviaremos el valor (ojo con tipos complejos, como las fechas, eque en PHP son objetos).
  - Tipo de Datos: En general, debe coincidir con el tipo que hemos definido para el parámetro dentro del informe iReport (tipos Java). Puede ser: Date, Time, String, Boolean, Int, Long, Double. Auqnue podemos apoyarnos en la potencia de las expresiones de iReport/jasperreport para "forzar" un tipo. En el apartado de Recomendaciones y generalidades sobre integración gvHidra y Jasper podemos encontrar más información al respecto.
- Línea 61: Guardamos la referencia al objeto en un atributo de la clase manejadora para poder acceder al mismo en la vista.
- Líneas 62-66: obtenemos un forward para el listado (una views que ahora luego veremos) y lo abrimos en una ventana nueva (método openWindow). Finalmente devolvemos el forward normal de la acción particular mostrando un mensaje en pantalla. Opcionamente, se puede crear el ActionForward('gvHidraNoAction') para que no recargue la ventana

De las últimas líneas comentadas anteriormente, deducimos que el fichero mappings.php debe contener el siguiente código:

```
...
$this->_AddMapping('claseDemo__iniciarVentana',
'TinvListFacturaJasper');
$this->_AddForward('claseDemo__iniciarVentana', 'gvHidraSuccess',
'index.php?view=views/listados/p_demoJasper.php');
$this->_AddForward('claseDemo__iniciarVentana', 'gvHidraError',
'index.php?view=igep/views/aplicacion.php');

$this->_AddMapping('claseDemo__abrirListado',
'TinvListFacturaJasper');
$this->_AddForward('claseDemo__abrirListado', 'correcto',
'index.php?view=views/listados/p_demoJasper.php');
$this->_AddForward('claseDemo__abrirListado', 'mostrarListado',
'index.php?view=views/listados/l_demoJasper.php');
...
```

Continuemos ahora con el último paso, la vista. Para integrar el listado generado con la ventana, hemos utilizado una llamada al forward mostrarListadoEn nuestro ejemplo l_facturaJasper.php

```
1: <?php
1: require_once('include/jasper/modulosPHP/informeJasper.php');
2:
```

```
3:
4:    if (IgepSession::existeVariable("claseDemo","infJasper_demo"))
9:    {
10:     $informeJ = &
 IgepSession::dameVariable("claseDemo","infJasper_demo");
11:     $informeJ->createResultFile('pdf');
12:   }
13:   IgepSession::borraVariable("claseDemo","lanzarInforme");
14:
15:
···SIGUE···
```

Del código anterior destacan:

- Linea 10: Recuperamos el atributo que referencia al listado de la sesión.
- Linea 11: Invocamos la creación del informe.

C.2. Depuración de errores al intentar ejecutar informes JasperReports en gvHidra

Para activar el modo de depuración de informes Jasper en GvHidra, dentro del proyecto PHP en el que estemos trabajando iremos al fichero: include/jasper/modulosPHP/informeJasper.PHP

Y dentro de el, hay que moverse a la linea 2 y cambiar la instrucción:

```
define('DEBUG', FALSE);
```

por:

```
define('DEBUG', TRUE);
```

y a partir de ese momento los listados ya no se exportarán a PDF ni se visualizarán y en su lugar se imprimirá en el navegador toda la información siguiente:

- comando que asigna todas las variables de entorno que requieren la máquina virtual de java y jasperreports para poder ejecutarse correctamente. En principio esto sólo será útil cuando se haya instalado la aplicación en un nuevo servidor web y queramos comprobar que la instalación ha sido correcta, ya que lo que se visualiza son todos los directorios en los que se va a buscar los ficheros jar's que contienen los ejecutables de jasperreports y otras librerias necesarias.
- comando que solicita la ejecución de la libreria Java de Adaptación de JasperReports a gvHidra, es decir: InformeJasper.class
- los resultados obtenidos, que variarán según haya sucedido un error o no y también según el momento en el que se haya producido el error.

En el caso de que el error esté arreglado y funcione todo bien se presentará:

- El fichero XML que intercambian InformeJasper.php y InformeJasper.class con la solicitud del informe a lanzar, que se compone de una primera parte en la que veremos el DTD que debe seguir el fichero xml y a continuación veremos los parámetros que se han pasado.

  Hay que revisar que todos los parámetros que el usuario introduzca se envien con el valor correcto. Existe la posibilidad de que el fichero XML no se haya podido generar y ese caso lo detectaremos fácilmente porque no se mostrará el fichero en el navegador.

En el caso de que el error haya surgido al intentar ejecutar el informe con los parámetros y la conexión pasada, que es el caso más habitual, se nos presentará un mensaje al final del navegador que dirá algo similar a:

```
   -informeJasper.php: No se generó el fichero resultado: /tmp/
listadoAlumnosJasper_XiAaZx.pdf
```

Y justo arriba de este mensaje aparecerá toda la pila de mensajes de error devuelta por JasperReports, por ejemplo si no pasamos el valor de un parámetro requerido obtendriamos un mensaje como éste:

```
Resultado Ejecución: net.sf.jasperreports.engine.JRException: Error
 executing SQL statement for :
listadoAlumnosJasper at
 net.sf.jasperreports.engine.query.JRJdbcQueryExecuter.createDatasource(JRJdbcQuer
at
 net.sf.jasperreports.engine.fill.JRFillDataset.createQueryDatasource(JRFillDatase
 at
net.sf.jasperreports.engine.fill.JRFillDataset.initDatasource(JRFillDataset.java:6
 at
net.sf.jasperreports.engine.fill.JRBaseFiller.setParameters(JRBaseFiller.java:892)
 at
net.sf.jasperreports.engine.fill.JRBaseFiller.fill(JRBaseFiller.java:716)
 at
net.sf.jasperreports.engine.fill.JRBaseFiller.fill(JRBaseFiller.java:669)
 at
net.sf.jasperreports.engine.fill.JRFiller.fillReport(JRFiller.java:63)
 at
net.sf.jasperreports.engine.JasperFillManager.fillReport(JasperFillManager.java:40
 at
net.sf.jasperreports.engine.JasperFillManager.fillReport(JasperFillManager.java:23
 at
InformeJasper.createResultFile(InformeJasper.java:282) at
InformeJasper.cargaFichDatosEntrada(InformeJasper.java:724) at
InformeJasper.main(InformeJasper.java:338) Caused by:
 org.postgresql.util.PSQLException:
No se ha especificado un valor para el parámetro 1. at
org.postgresql.core.v3.SimpleParameterList.checkAllParametersSet(SimpleParameterLi
 at
org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:182)
 at
```

```
org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:45
 at
org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statemen
 at
org.postgresql.jdbc2.AbstractJdbc2Statement.executeQuery(AbstractJdbc2Statement.ja
 at
net.sf.jasperreports.engine.query.JRJdbcQueryExecuter.createDatasource(JRJdbcQuery
11 more
```

Nótese que en primer lugar encontramos todos los métodos del API de JasperReports que han fallado empezado por el primer método que haya abortado su ejecución, y que debe acabar llegando a la clase InformeJasper.class A continuación aparece un mensaje "Caused by:" que es la linea más importante de todo el listado ya que en ella JasperReports intentará explicarnos la causa del error .

Seguiremos intentado arregalr el error hasta que al lanzarlo en el navegador ya no aparezca la pila de mensajes de error y tampoco aparezca el aviso de:

```
-informeJasper.php: No se generó el fichero resultado:
```

Si lo que tenemos en pantalla es únicamente es el fichero XML que intercambian InformeJasper.php y InformeJasper.class es que ya hemos corregido el error y en este caso debemos abandonar el modo de depuración volviendo a colocar en el fichero include/jasper/modulosPHP/informeJasper.PHP la linea define('DEBUG', FALSE);

C.3. Recomendaciones y generalidades sobre integración gvHidra y Jasper

Los parámetros en Jasper Reports se pueden dar de alta desde:

- Menu Ver / Parámetros del Informe / Botón Nuevo
- Panel de 'Librería', seleccionamos Parámetros y con botón derecho "Nuevo Parámetro"

Los parámetros de Jasper, pueden ser de 2 tipos, siendo referenciados dentro del informe Jasper con un formato distinto para cada tipo.

A los parámetros les podemos especificar un nombre, tipo y un valor por defecto. Si marcamos el check "use as prompt", cuando ejecutemos el informe desde dentro del entorno iReports, nos pedirá que le demos un valor cada vez que lo ejecutemos, dándonos también la opción de asignarle al parámetro el valor por defecto que hayamos especificado.

- FORMATO A: $P{Nombre_Parámetro}
  - Este parámetro podrá ser utilizado tanto en la Select, como en cualquier variable calculada o en las diferentes expresiones Java que Jasper permite introducir.
  - Las expresiones en las que se puede utilizar son:
    - en las propiedades de los campos que se colocan en el informe, en concreto son "Print When expression" y "text Field expression"

- en la expresión "Print When expression" de las bandas de las que consta el informe

- La ventaja de este sistema es que como el parámetro tiene un tipo, cualquier expresión Java que utilice el parámetro podrá ser comprobada por Jasper cuando se compile el informe, y por tanto se nos informará de todos los errores en tiempo de compilación.

Se pueden crear tantos parámetros de tipo $P{Nombre_Parámetro} como se necesiten en el informe.

En muchas ocasiones la aplicación PHP presentará una pantalla de selección de datos en la que NO será obligatorio especificar todos los campos (que se pasan al informe Jasper como parámetros).

Cuando en esta pantalla de selección de datos se deja en blanco un campo, se entiende que se solicitan todos los registros, contengan lo que contengan.

Para estos parámetros opcionales existe la posibilidad de escribir el WHERE de la select de la forma siguiente, suponiendo que "titulo" es un parametro opcional y su valor por defecto se ha fijado a "null": tddm_expediente.codtit = COALESCE($P{titulo},tddm_expediente.codtit)

Este "truco" NO funcionará bien cuando la columna en cuestión (tddm_expediente.codtit) admita el valor NULL, en cuyo caso NO se debe utilizar este sistema.

El método anterior se basa en que el parámetro $P{titulo} tiene valor por defecto "null", por lo que cuando no se pasa $P{titulo} se queda a null y como la función SQL COALESCE devuelve el primer valor no nulo, devolverá el segundo parámetro reduciéndose la condición a tddm_expediente.codtit = tddm_expediente.codtit, que es una operación trivial que devuelve TRUE... excepto cuando el valor de tddm_expediente.codtit sea NULL, por que sql considera que (NULL = NULL) NO devuelve Verdadero (sino que devuelve NULL), el resultado es que aquellas filas que contengan NULL se pierden porque no pueden satisfacer la condición.

La solución en este caso pasa por utilizar la otra forma alternativa de parámetros que tiene JasperReports, el formato B.

- FORMATO B: $P!{Nombre_Parámetro}

  Este parámetro se da de alta como un parámetro normal, pero deberá tener siempre el tipo de valor "java.lang.String" y el valor por defecto a "". Lo que cambia es la forma de referenciarlo en el informe que será siempre como $P!{Nombre_Parámetro}.

Los parámetros $P!{Nombre_Parámetro} se interpretan antes que los parámetros $P{Nombre_Parámetro}, por tanto, en teoría, es posible que la cadena que pasemos como valor de $P!{Nombre_Parámetro} contenga a su vez expresiones que deban ser posteriormente calculadas o pasadas desde otra fuente. Se supone que podríamos llegar a incluir un parámetro de formato A $P{Nombre_Parámetro} dentro del valor que pasemos a una cadena de formato B $P!{Nombre_Parámetro}, si bien, esto no es útil en gvHidra ya que toda la información de selección de datos se introduce en una misma pantalla previa al lanzamiento del informe, aunque puede serlo en otros contextos.

Los parámetros $P!{Nombre_Parámetro} permiten modificar la cadena SQL que sirve de base para el informe. En concreto pueden contener:

1. La totalidad de la sentencia SQL en la que se basa el informe: en ese caso el informe solo contiene en su clausula base: $P!{CadSQL}

2. Una clausula completa de las que componene una sentencia SQL, como por ejemplo "ORDER BY 1,2" o "WHERE tddm_expediente.codtit='PAYA"

3. Una parte de alguna clausula de las que componene una sentencia SQL, por ejemplo (dentro de la clausula WHERE) " AND tddm_expediente.codtit='PAYA' ". Para utilizar esta última solución, en la sentencia SQL debemos añadir un parámetro $P!{CadenaWhereOpcional} al final de la claúsula WHERE:

   - cuando el usuario no especifique un valor para el campo opcional "codigo titulo", el parámetro $P!{CadenaWhereOpcional} se pasará como cadena vacía

   - cuando el usuario rellene por ejemplo el parámetro opcional "codigo titulo" con el valor "PAYA" el parámetro $P!{CadenaWhereOpcional} tomará el valor "AND tddm_expediente.codtit='PAYA' "

La solución 1, ofrece la ventaja de que toda la sentencia sql puede venir de nuestra aplicación PHP, esto es útil cuando el programa también deba utilizar la sentencia SQL para por ejemplo, comprobar que existe algún registro con las condiciones introducidas. Basta con que la versión de la sentencia SQL que se utiliza en la aplicación PHP añada un "LIMIT 1" para que sólo se intente recuperar 1 registro como máximo, y en caso de que aparezca 1 registro, se pasa la cadena sin el LIMIT 1 al informe.

Esta técnica ESTÁ RECOMENDADA para los casos en los que la aplicación compruebe que hay registros que cumplen las condiciones porque facilita el mantenimiento de las consultas, ya que si hay algún cambio o corrección posterior de algún error, la consulta se modificará sólo en un sitio, y no habrá inconsistencias entre la consulta de la aplicación PHP y la de JasperReports.

Cuando no se especifica la consulta del informe, JasperReports sólo comprobará la cadena SQL en tiempo de ejecución cuando la reciba y además iReports no puede recuperar automáticamente los campos disponibles para diseñar el informe. Pero esto se puede conseguir de dos formas:

- Desactivando la opción de "Recuperación automática de los campos", luego copiando toda la select, (ATENCIÓN: iReports la va a ejecutar, por lo que debemos filtrar por algun campo en la clausula WHERE o añadir al final un LIMIT 1 para que NO se descargue toda la Base de Datos !! Puesto que si se baja demasiados datos se nos puede quedar sin memoria o estar mucho rato sin respondernos) y finalmente pulsando el botón de "Leer Campos" lo que creará automáticamente todos los campos devueltos por la select.

- Creando manualmente todos los campos, con sus respectivos tipos. Recordad que Java distingue las mayúsculas de las minúsculas por lo que hay que ceñirse a los nombre de los campos que devuelva la select que vamos a pasar

En el caso de que el mantenimiento sea de tipo Maestro/Detalle, si el listado se lanza desde una ventana que tiene parámetros opcionales, se puede aplicar al informe Maestro todas las soluciones anteriores para pasarle los parámetros opcionales, en concreto para los casos en los que se compruebe que existen registros que cumplan las condiciones introducidas en la pantalla de selección, se recomienda utilizar un parámetro con el FORMATO B: $P!{Nombre_Parámetro} y con la solución A, mientras que el Detalle puede llevar la SQL en el propio informe, ya que es la consulta Maestra la que decide qué registros se van a mostrar.

Apéndice D. Instalación rápida del entorno

D.1. Introducción

En este documento pretendemos dar una guía para poder tener un entorno de trabajo para gvHIDRA. Tenemos una serie de requisitos y dependencias que nuestro sistema tiene que cumplir para poder ejecutar aplicaciones con el framework y aquí intentaremos hacer referencia a todas ellas.

Si se es conocedor de las instalaciones de entornos web, simplemente tiene que atender a las recomendaciones del apartado Requisitos. Si, en cambio, es la primera experiencia con estos entornos, o no está familiarizado con su instalación, le recomendamos que haga uso de la guía paso a paso.

D.2. Requisitos

Los requisitos para instalar un entorno gvHIDRA son:

- **Servidor Web**: Necesitamos instalar un servidor web que pueda trabajar con PHP. Recomendamos Apache.
- **PHP**: Instalaremos una versión PHP 5. Las versiones de gvHIDRA 4.0.0 y posteriores ya son compatibles con las versiones 5.3 y 5.4 de PHP.
- **Librerías PEAR**: Internamente, el framework hace uso de librerías propias de PEAR. Concretamente tenemos que instalar:
  - PEAR
  - MDB2 (si se quieren utilizar conexiones MDB2). Desde la versión 4.0.9 se pueden utilizar los drivers PDO.
  - MDB2_Driver_Mysql (si quiere conectar con otro SGBD instale el driver apropiado).
  - Mail,Net_SMTP (sólo si se quiere utilizar la clase de envío correo)

  En versiones anteriores, es necesario incluir la libería PEAR::Auth para realizar la validación. Desde la versión 4.1.0, ya no es necesario instalarla al estar integrada.

D.3. Instalación entorno rápido

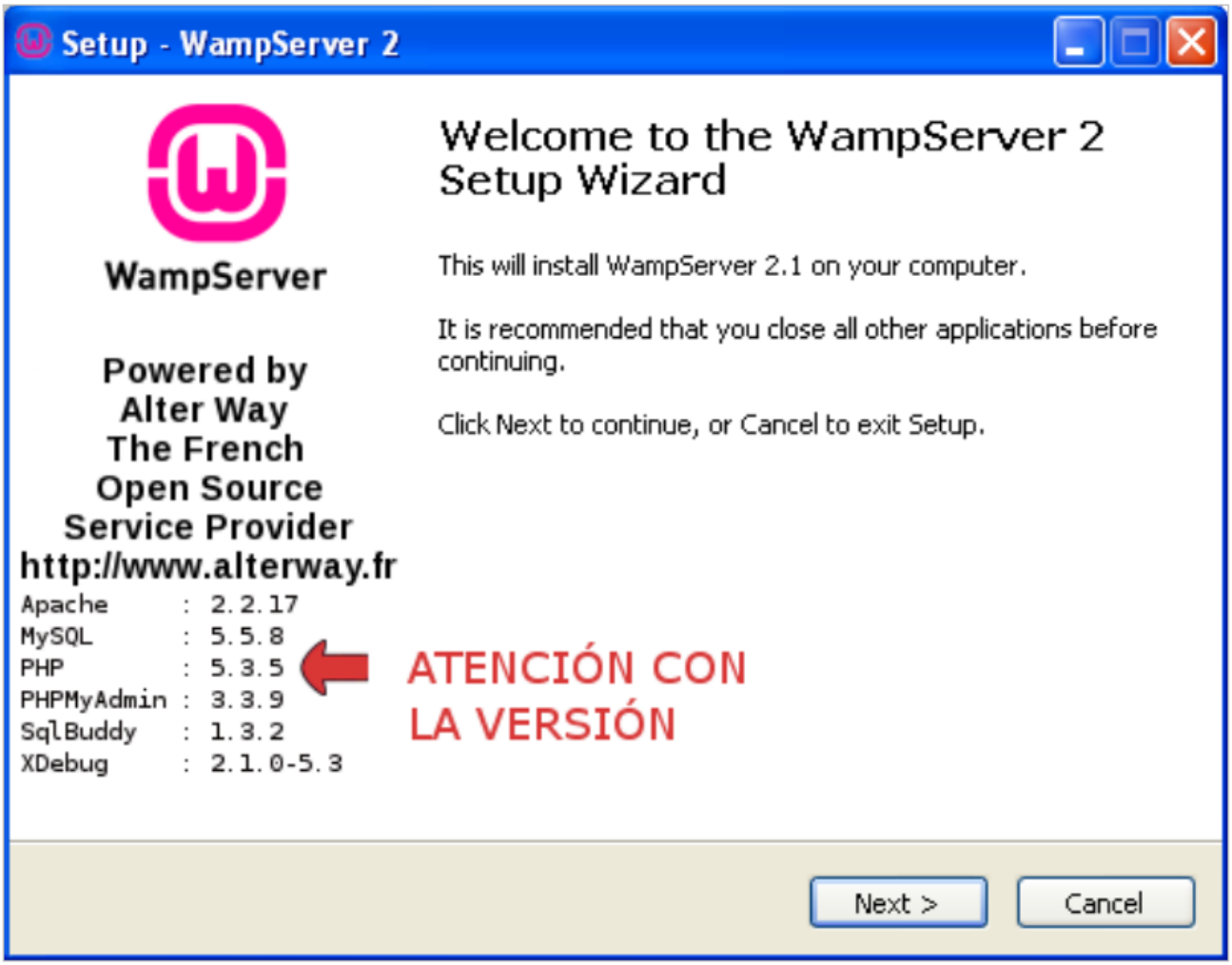
Evidentemente, la mayor dependencia la marca el sistema operativo que utilicemos en nuestro servidor/máquina. Por ello, vamos a dividir la guía en instalación para Microsoft Windows e instalación para entornos Linux.

- **MICROSOFT WINDOWS**

  Para facilitar el trabajo, al tratarse de una guía rápida y sencilla, vamos a hacer uso de uno de los paquetes WAMP disponibles en la web; concretamente el WAMPSERVER (http://www.wampserver.com/en/). Se puede hacer uso de cualquiera, siempre que se tengan en cuenta los requisitos anteriores.

Primer paso, descargar el wamp. Para ello accedemos a la página del proyecto y descargamos la última versión.

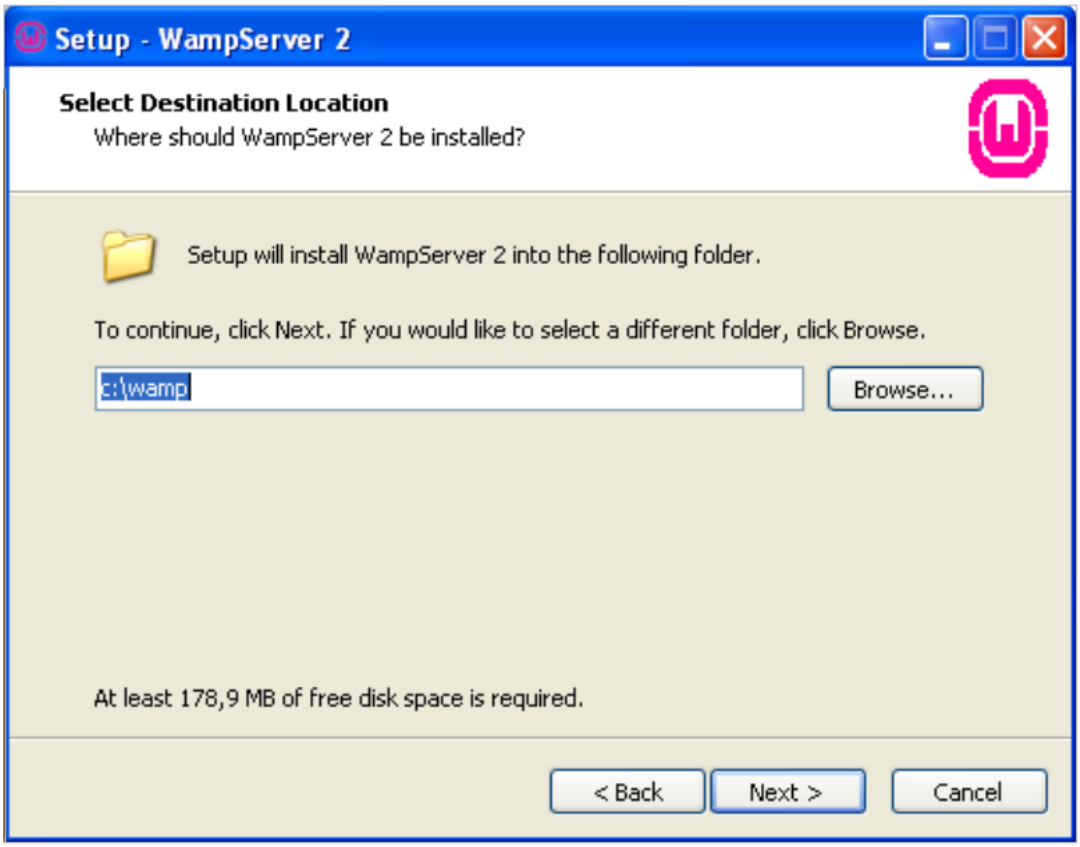
Al ejecutar el fichero que hemos bajado, nos aparecerá un wizard de instalación con una ventana similar a esta:



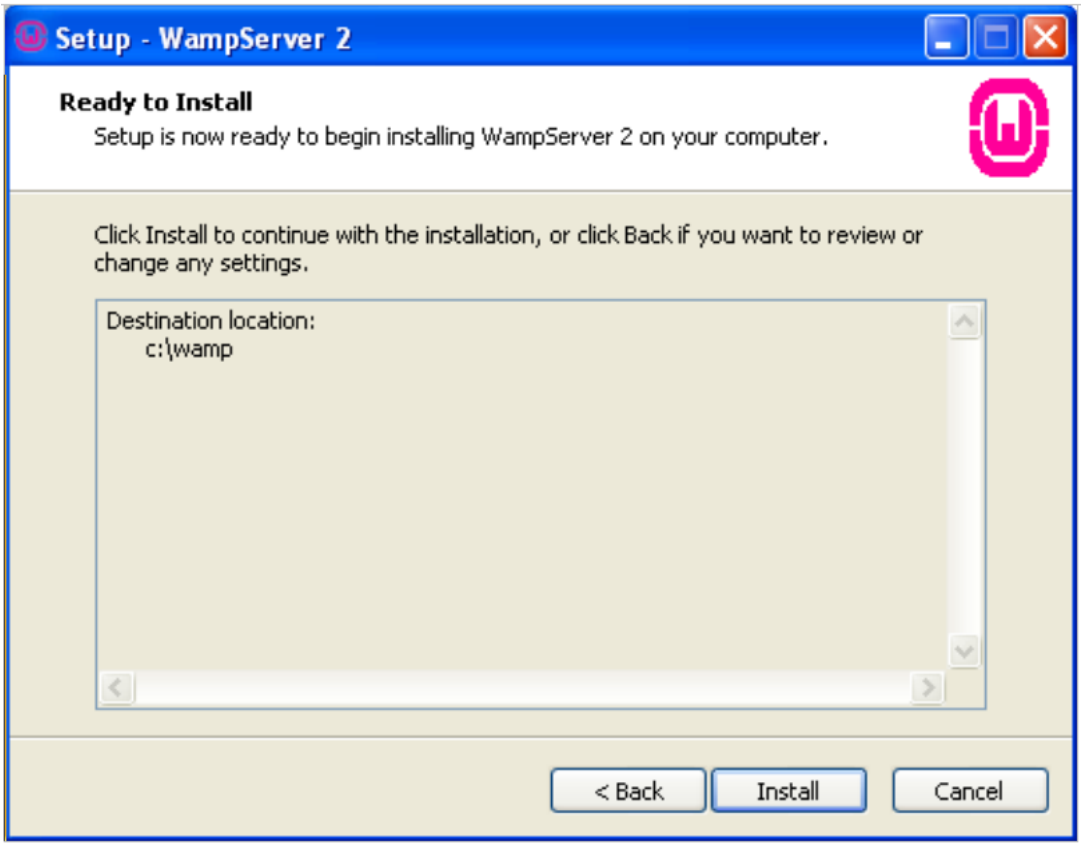
Nota: es importante tener en cuenta si la versión que se instala es compatible o no con el framework. Si no lo es, tendremos que realizar tras esta instalación, el paso 3 de este manual.

Al pulsar Next, accederemos a un documento de licencia, donde explica la licencia GPL del proyecto. Pulsamos Next.

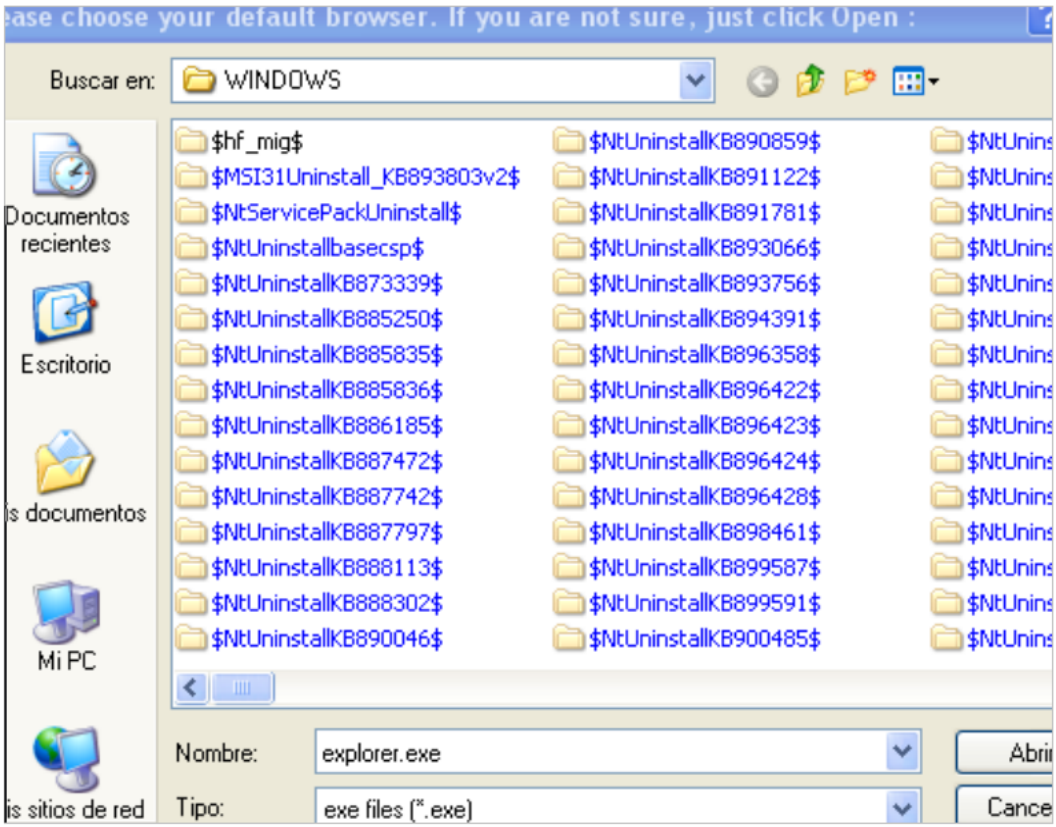
En el siguiente apartado, nos solicita la ruta de instalación del wamp. Aceptamos la ubicación propuesta (en caso de cambiarla, se debe tener en cuenta en los siguientes pasos).



Tras configuraciones propias del sistema operativo Windows, el wizard pasa a solicitarnos la confirmación. Pulsamos install.



Finalmente, nos solicita la ubicación del navegador web que queramos utlizar. Por defecto aparece el Explorer, nosotros sugerimos la utilización del Firefox. Seleccione la ubicación del binario correspondiente.



Una vez acabada la instalación, el servidor intentará arrancarse. Si tiene activado el firewall, es posible que salte un mensaje de advertencia como este:

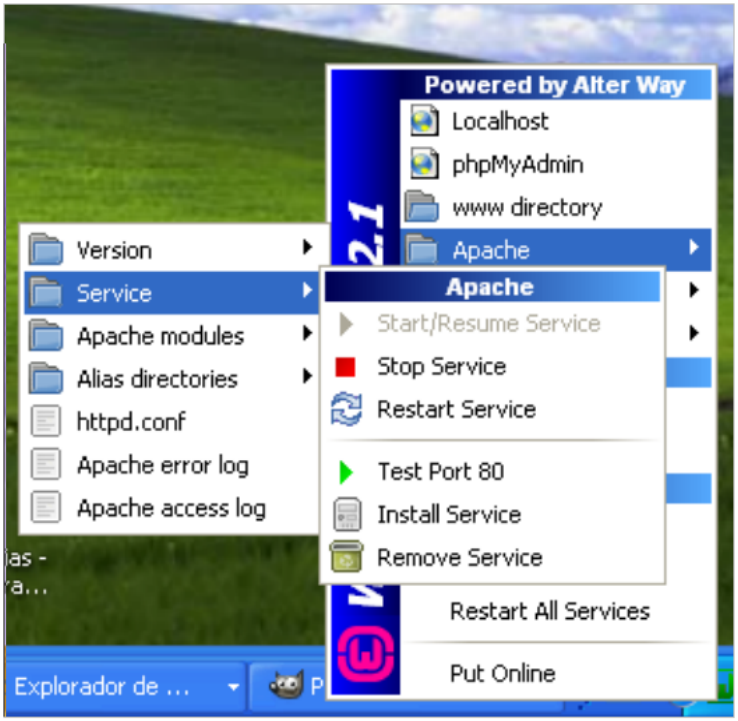
Desbloquee el Apache y, si todo ha ido bien, aparecerá en la bandeja de sistema un icono como este

Si el icono **no está en color verde**, eso se debe a que no ha conseguido arrancar alguno de los servicios.

Según el color del icono sabremos qué servicio ha fallado:

- *Color rojo*.

  No ha podido arrancar el apache. Acceda al menú del wampserver en el apartado Apache/Services.



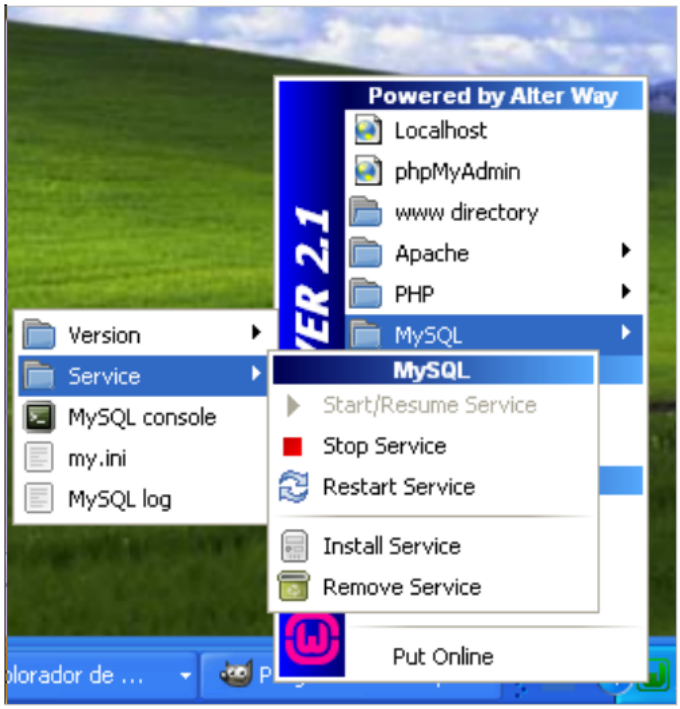
  1. Compruebe que el puerto 80 está libre con Test Port 80. Si no está libre, consulte la documentación de Apache para configurar otro puerto de trabajo.

  2. Instale el servicio mediante la opción Install Service.

  3. Pulse Start/Resume Service

- ***Color naranja***.

  No ha podido arrancar mySQL. Acceda al menú MySQL/Service.

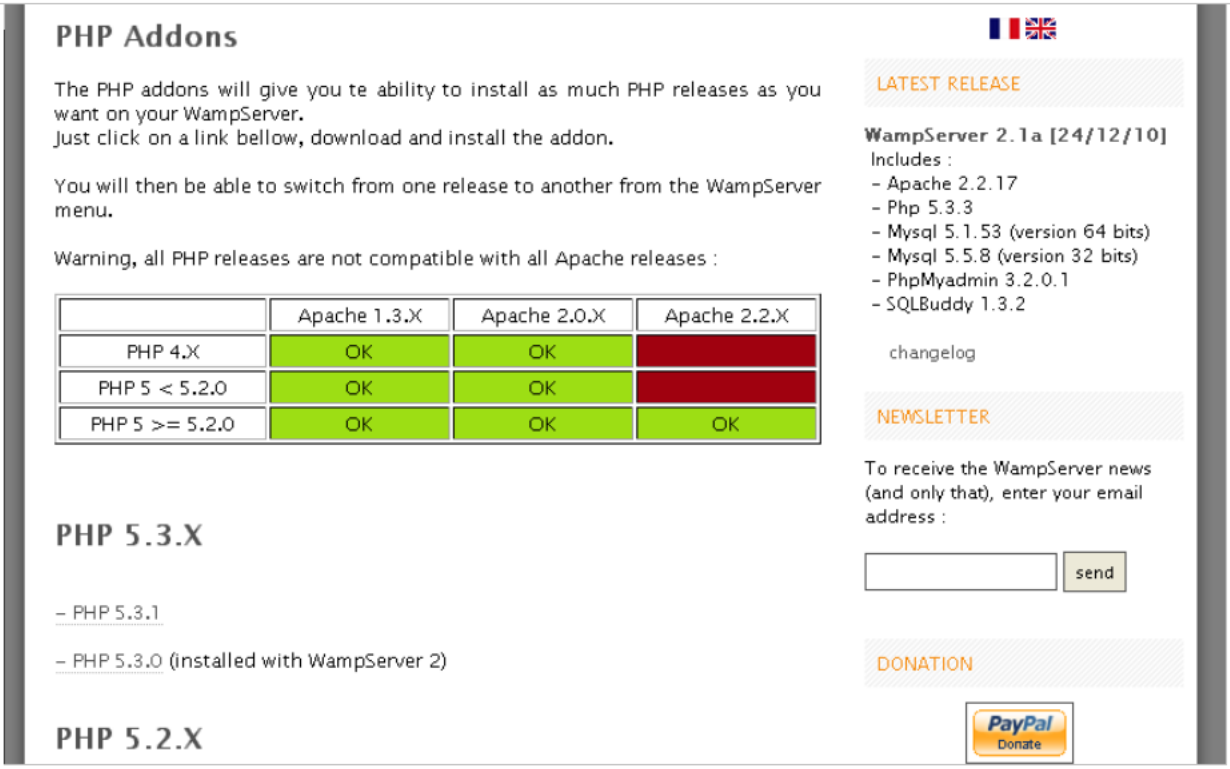


1. 1.Instale el servicio con Install Service

2. 1.Arranque el servicio con Stara/Resume Service

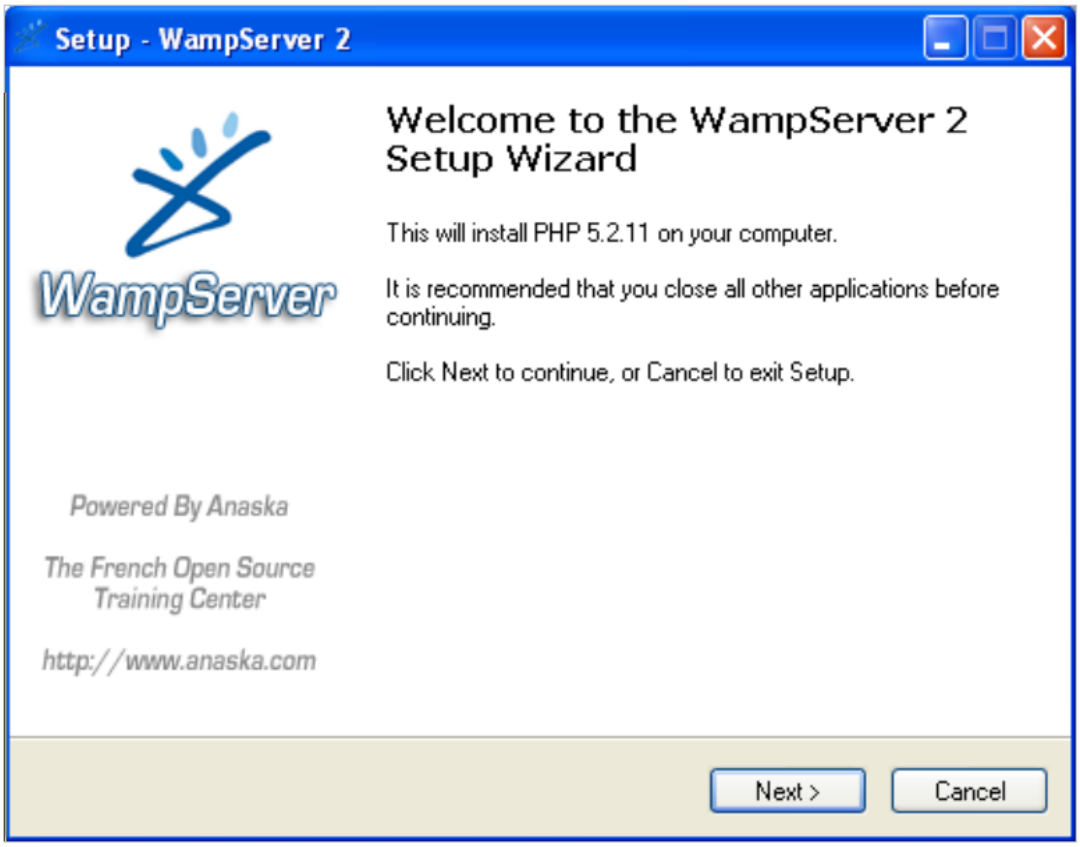
- **INSTALACIÓN DE VERSIONES ANTERIORES DE PHP. INSTALACIÓN DE ADDONS**

Si lanzas este punto es porque la versión de PHP que contiene el paquete instalado no es compatible con la versión del framework (las versiones anteriores a la 4.0.0 sólo eran compatibles con PHP 5.2.X). Para solucionar el problema, tenemos que instalar un addons. Para ello, en la página del proyecto, pulsamos en ADDONS y luego en el enlace PHP. Nos aparecerá algo parecido a esto:

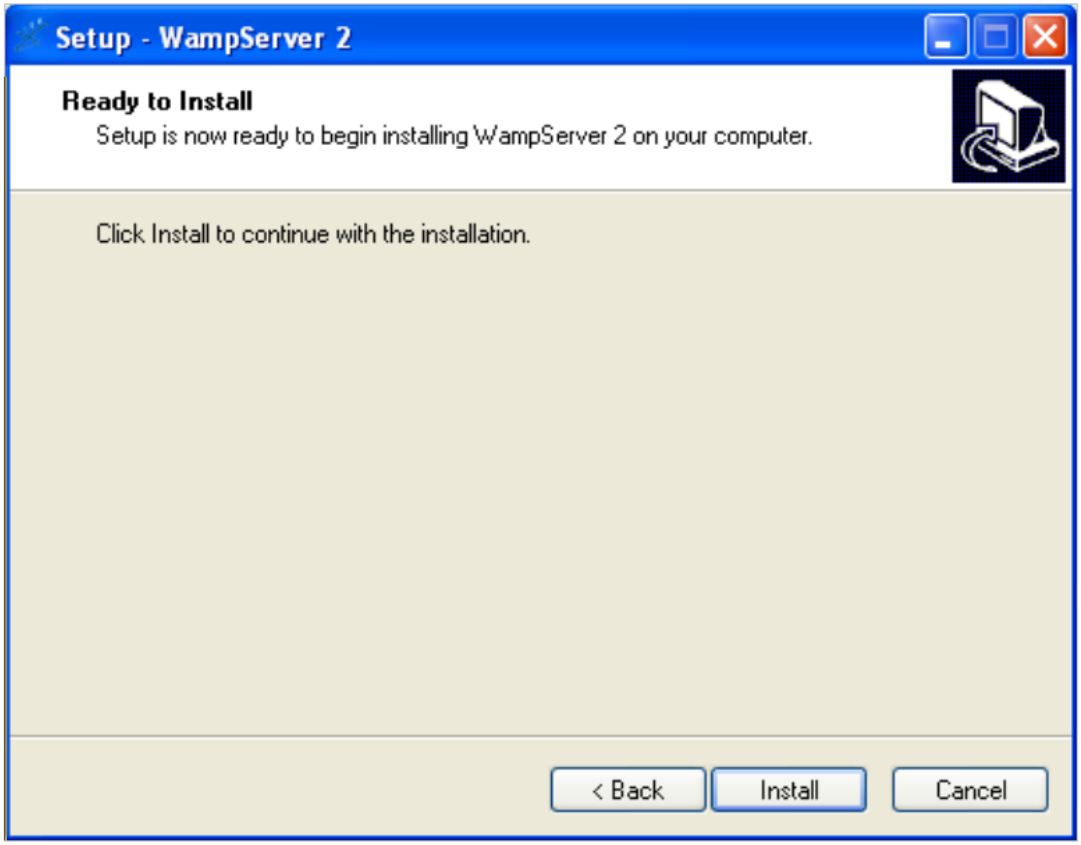


Aquí seleccionamos la versión deseada, en nuestro ejemplo PHP-5_2_11 y guardamos el archivo.

Ejecutamos el archivo y (tras dar permiso si nos ha aparecido alguna alerta) se nos abrirá una ventana con un wizard de instalación.

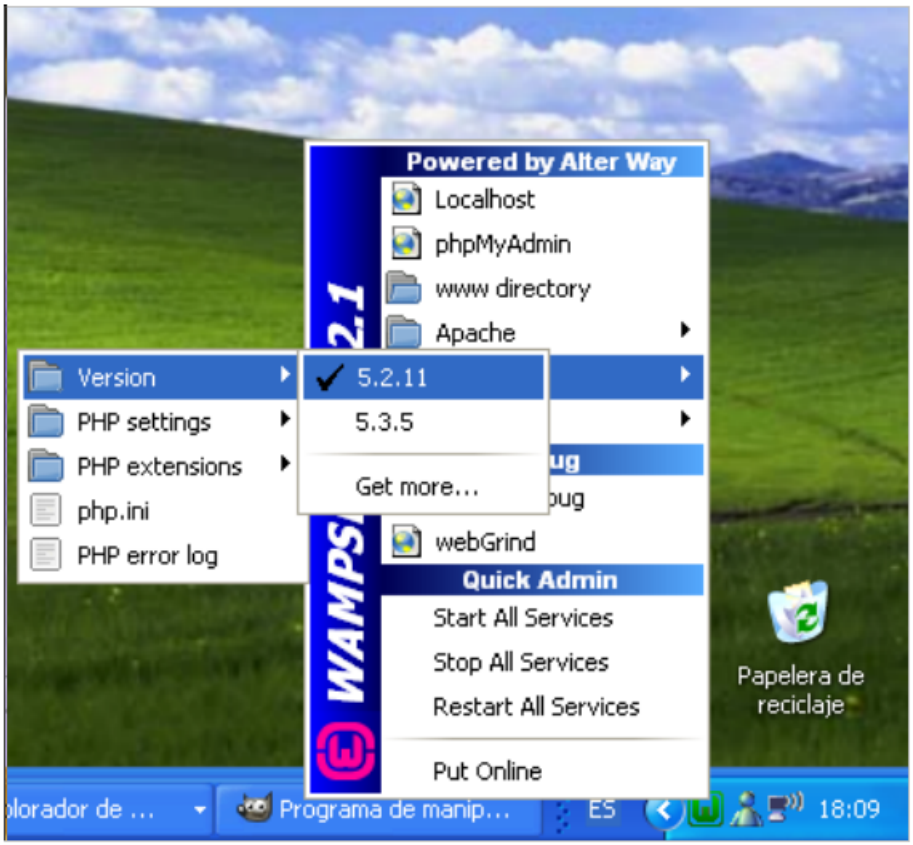


Le damos a siguiente y luego a install:



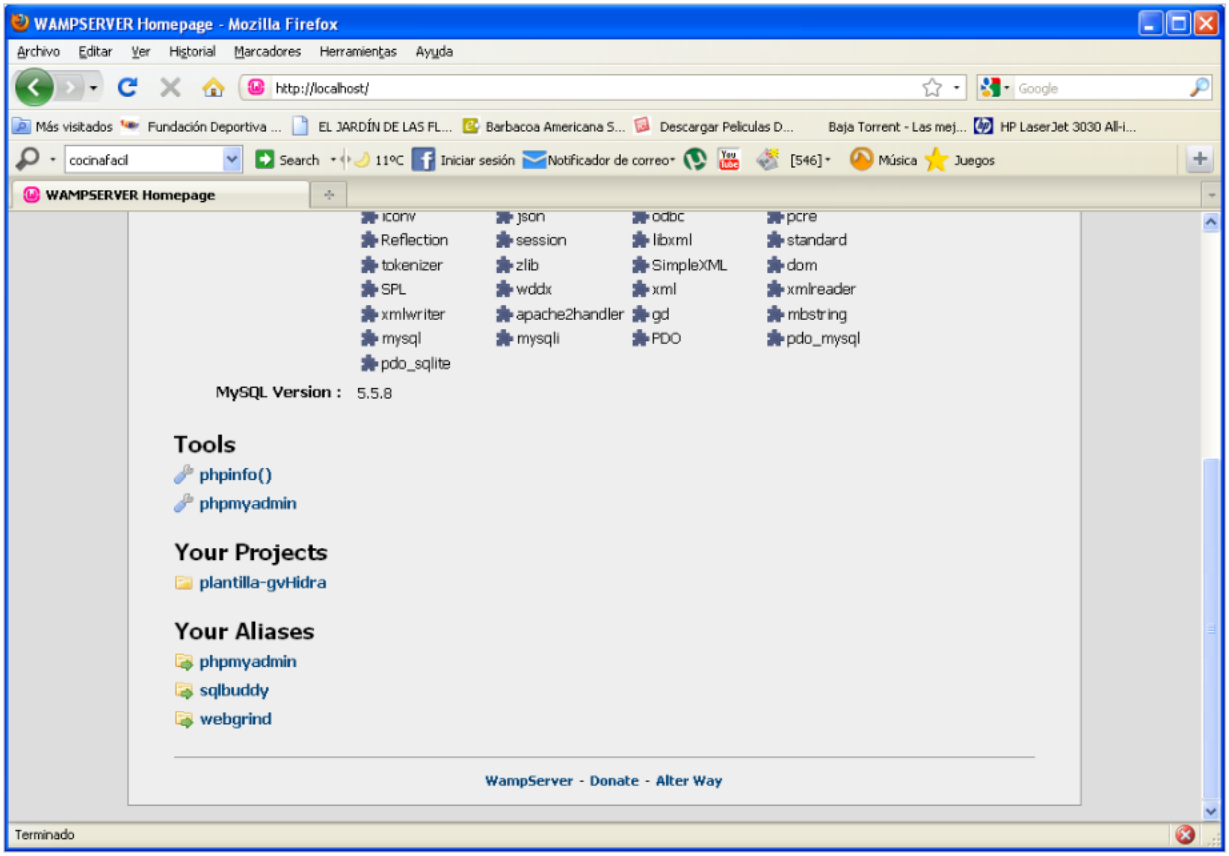
Nos preguntará de nuevo por la dirección del servidor SMTP y la dirección de correo.

Una vez finalizada la instalación, debemos indicarle al wampserver que queremos utilizar la versión del php adecuada. Para ello accedemos al menú de proyecto PHP/Version.



Seleccionamos la versión adecuada y reiniciamos el servidor (Restart all sevices).

Para comprobar que todo ha funcionado correctamente, desde un navegador deberemos ver una página como la siguiente al acceder a la ruta http://localhost.



- **LIBRERÍAS PEAR**

Una vez el entorno instalado, necesitamos añadir las librerías PEAR necesarias para que el framework funcione. Dependiendo de la versión que tengamos instalada de PHP, algunos pasos pueden cambiar.

El cambio más importante está en la primera parte de la instalación. Las últimas versiones de PHP no distribuyen el fichero go-pear por lo que deberemos descargarlo.

Nota: para este paso se requiere conexión a internet

Abrimos una consola y nos ubicamos en el directorio de php (sustituir X.X por la versión correspondiente):

```
cd C:\wamp\bin\php\php5.x.x
```

Una vez en él, tenemos que comprobar si existe el fichero go-pear.bat o go-pear.phar. Si no disponemos de él, debemos decargarlo de la siguiente ubicación http://pear.php.net/go-pear.phar

Tras descargar el archivo y ubicarlo en la carpeta de nuestro php, lanzaremos el comando go-pear dependiendo de la extensión del fichero:

```
php go-pear.phar
o
go-pear.bat
```

Una vez acabada la instalación, comprobamos que todo a funcionado correctamente. Ejecutamos:

```
pear list

INSTALLED PACKAGES, CHANNEL PEAR.PHP.NET:
=========================================
PACKAGE          VERSION STATE
Archive_Tar      1.3.3   stable
Console_Getopt   1.2.3   stable
PEAR             1.9.0   stable
Structures_Graph 1.0.2   stable
XML_Util         1.2.1   stable
```

Una vez tenemos PEAR, vamos a instalar las librerías que necesitamos:

```
pear install MDB2
pear install MDB2_Driver_mysqli
pear install MAIL
pear install Net_SMTP
```

Nota: si se quiere utilizar otro SGBD distinto de MySQL tendremos que instalar el driver correspondiente del MDB2.

```
pear list

INSTALLED PACKAGES, CHANNEL PEAR.PHP.NET:
=========================================
PACKAGE            VERSION STATE
Archive_Tar        1.3.3   stable
Console_Getopt     1.2.3   stable
MDB2               2.4.1   stable
MDB2_Driver_mysqli 1.4.1   stable
Mail               1.2.0   stable
Net_SMTP           1.5.0   stable
Net_Socket         1.0.10  stable
PEAR               1.9.0   stable
Structures_Graph   1.0.2   stable
XML_Util           1.2.1   stable
```

Nota: Recuerde, para acceder a SGBD que no sean MySQL deberá también habilitar las extensiones pertinentes.

Para finalizar, para evitar problemas de configuración, tenemos que asegurarnos de que las nuevas clases pear sean accesibles por el php. Para ello accederemos al fichero a través del m enu del wampserver/php->php.ini y comprobaremos que el include_path es:

```
include_path = ".;c:/wamp/bin/php/php5.X.X/PEAR;"
```

Finalmente, fijamos el valor de la variable de comunicación de errores (error_reporting). En el mismo fichero php.ini, podemos acceder a ella y consultar las opciones disponibles. Por ejemplo:

```
error_reporting  =  E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR
```

Reiniciamos el servidor (Reestart all Services) y ¡ya tenemos el entorno preparado para gvHIDRA!. Vamos a probar, descarga la última versión de gvHIDRA y la descomprimimos en nuestro directorio www (generalmente c:\wamp\www, consultar manager para su verificar su ubicación). Una vez descomprimido, seguimos los pasos de instalación que allí se nos indican (guia_rapida.txt).

Nota: recuerda quitar el solo lectura al templates_c

- **RECOMENDACIÓN: INSTALACIÓN DEL LOG**

Aunque no es necesario, es conveniente tener en cuenta la configuración del log en nuestro nuevo entorno. Aquí os proponemos una configuración:

```
-- Debugger gvHIDRA
-- Tiempo de generacion: 10-02-2011 a las 04:56:16
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
CREATE DATABASE `debugger` DEFAULT CHARACTER SET latin1 COLLATE
 latin1_spanish_ci;
USE `debugger`;
--
-- Base de datos: `debugger`
--
-- --------------------------------------------------------
--
-- Estructura de tabla para la tabla `tcmn_errlog`
--
DROP TABLE IF EXISTS `tcmn_errlog`;
CREATE TABLE `tcmn_errlog` (
  `iderror` int(11) NOT NULL auto_increment,
  `aplicacion` varchar(30) NOT NULL default '',
  `modulo` varchar(10) default NULL,
  `version` varchar(10) NOT NULL default '',
  `usuario` varchar(30) NOT NULL default '',
  `fecha` timestamp NOT NULL default CURRENT_TIMESTAMP on update
 CURRENT_TIMESTAMP,
  `tipo` varchar(10) NOT NULL default '',
  `mensaje` text NOT NULL,
  PRIMARY KEY  (`iderror`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Log debugger'
 AUTO_INCREMENT=1;
```

- **GNU/LINUX**

El caso de Linux requiere un conocimiento más amplio que en Windows. En este documento, vamos a dar unas pautas de instalación; pero puede que no den el resultado esperado ya que algunas herramientas dependen de la distribución seleccionada. Vamos a dar dos ejemplos de instalación.

- **OPCIÓN DEBIAN/UBUNTU**

Desde consola ejecutar como root:

```
isantos:~# apt-get install php5 php5-gd php-pear libapache2-mod-php5 php-auth
#Instalamos mysql
isantos:~# apt-get install mysql-server

#Reiniciamos apache para asegurarnos de que carga correctamente los módulos de
 php y pear.
isantos:~# service /etc/init.d/apache2 restart
```

- **OPCIÓN COMPILACIÓN DE FUENTES**

El primer paso es instalar Apache, Mysql y todos los paquetes de desarrollo que necesitaremos para compilar PHP. Cabe aclarar que según los parámetros que le pasemos al configure podemos necesitar mas paquetes. Para este ejemplo haremos la instalación mas simple solo con soporte de Mysql.

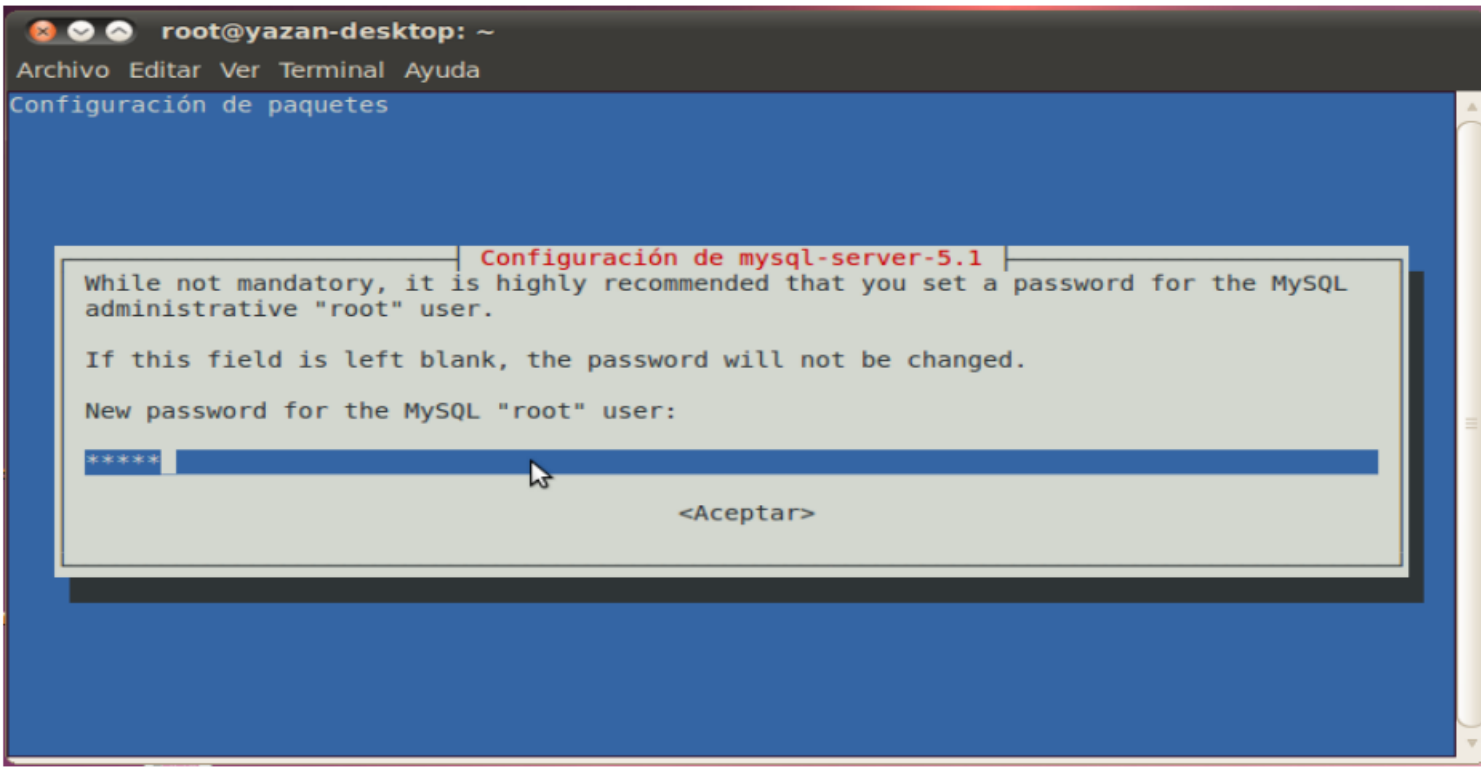
Primero cambiamos a usuario root:

```
sudo su -
```

1. Instalación de Apache y Mysql y con las librerías necesarias:

```
apt-get install flex apache2-threaded-dev libxml2-dev apache2 apache2-mpm-
prefork apache2-threaded-dev apache2-utils apache2.2-bin apache2.2-common
 libapache2-mod-auth-mysql mysql-server-5.1
```

Se nos pedirá una contraseña nueva para el root de mysql como se muestra en la imagen, (en el ejemplo "mysql").



2. Descargamos y descomprimimos la versión de PHP 5.4.X:

```
root@gvhidra-test:/instalaciones# wget http://es.php.net/distributions/
php-5.4.7.tar.bz2
root@gvhidra-test:/instalaciones# tar xvjf php-5.4.7.tar.bz2
```

3. Compilamos e instalamos PHP 5.4.X (Verificar que no se produce ningún error durante la compilación e instalación):

```
root@gvhidra-test:/instalaciones/php-5.4.7# ./configure --prefix=/usr/local/
php-5.4.7 --with-apxs2=/usr/bin/apxs2  --with-config-file-path=/usr/local/
```

```
php-5.4.7 --with-libxml-dir=/usr/include/libxml2 --with-openssl=/usr --with-
gd --with-jpeg-dir=/usr --with-png-dir=/usr  --enable-soap  --with-mysqli --
with-mysql=/usr/include/mysql

Opcional:
--with-pgsql=/ruta/instalacion/postgresql/
--with-oci8=shared,instantclient,/ruta/instalacion/oracle/instantclient


root@gvhidra-test:/instalaciones/php-5.4.7# make
root@gvhidra-test:/instalaciones/php-5.4.7# make install
```

- **LIBRERÍAS PEAR**

Una vez el entorno instalado, necesitamos añadir las librerías PEAR necesarias para que el framework funcione.

Nota: para este paso se requiere conexión a internet

Abrimos una consola y ejecutamos:

```
cd /usr/local/php/php_5.4.7/bin/
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear upgrade PEAR
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear channel-update -f
 "pear.php.net"
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear install --alldeps
 MDB2_Driver_mysqli
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear install --alldeps Mail
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear config-set preferred_state
 alpha
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear install --alldeps SOAP
root@gvhidra-test:/usr/local/php-5.4.7/bin# ./pear install --alldeps
 Image_graph
```

Nota: Recuerde, para acceder a SGBD que no sean MySQL deberá también habilitar las extensiones pertinentes.

Aunque no es obligatorio, es recomendable que en este paso configuremos la variable de comunicación de errores (error_reporting). En el mismo fichero php.ini, podemos acceder a ella y consultar las opciones disponibles. Proponemos darle el valor:

```
error_reporting = E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED
```

Reiniciamos el servidor (sudo /etc/init.d/apache2 restart) y ¡ya tenemos el entorno preparado para gvHIDRA!. Vamos a probar, descarga la última versión de gvHIDRA y la descomprimimos en nuestro directorio www (generalmente “/var/www/htdocs”). Una vez descomprimida, seguimos los pasos de instalación que allí se nos indican (guia_rapida.txt).