



Manual Usuario gvHidra

Versión 5.0.0

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.	7
1.2.1. Hardware.	7
1.2.2. Software.	7
1.3. Entendiendo el entorno.	7
1.4. Instalación del entorno.	9
1.4.1. ¿El entorno funciona correctamente?	9
1.5. Creando mi “hola mundo” en gvHidra.	11
2. Conceptos técnicos de gvHidra.	14
2.1. Arquitectura	14
2.1.1. Arquitectura por capas.	14
2.1.2. Configuración.	15
2.2. Aspecto visual.	15
2.2.1. Anatomía de una ventana gvHidra.	15
2.2.2. Modos de trabajo y ventanas gvHidra.	16
2.2.3. Patrones de interfaz.	17
2.3. Lógica de negocio.	21
2.3.1. Acciones y operaciones.	21
II. Elementos de gvHidra	26
3. Elementos básicos.	28
3.1. Estructura de aplicación.	28
3.1.1. Configuración estática: <i>gvHidraConfig.xml</i>	28
3.1.2. Configuración dinámica: <i>AppMainWindow</i>	33
3.1.3. Recomendaciones.	34
3.2. Breve guía para crear una pantalla	37
3.2.1. Introducción	37
3.2.2. Genaro: Generación automática en gvHidra	37
3.2.3. Revisión de un mantenimiento	42
3.3. Menú de una aplicación	49
3.3.1. Funcionamiento del menú	49
3.3.2. Opciones predefinidas para el menu	54
3.3.3. Autenticación y autorización (módulos y roles)	55
3.4. Diseño de pantalla con Smarty/plugins	58
3.4.1. ¿Qué es un template?	58
3.4.2. Cómo realizar una plantilla (tpl) básica.	58
3.4.3. Diseño avanzado de plantillas.	61
3.4.4. Documentación de los plugins	62
3.5. Código de la lógica de negocio	62
3.5.1. Operaciones y métodos virtuales	62
3.5.2. Uso del panel de búsqueda	72
3.5.3. Acciones no genéricas	75
3.5.4. Acciones de interfaz	78
3.6. Personalizando el estilo	81
3.7. Tratamiento de tipos de datos.	82
3.7.1. Características generales	82
3.7.2. Cadenas (gvHidraString)	83
3.7.3. Fechas	84
3.7.4. Números	88
3.7.5. Creación de nuevos tipos de datos	89

3.8. Listas de datos	90
3.8.1. Listas	90
3.8.2. Checkbox.	95
3.9. Mensajes y Errores.	96
3.9.1. Invocación desde código.	98
3.9.2. Invocación como confirmación.	98
3.10. Uso de datos por defecto.	99
4. Elementos de pantalla avanzados	100
4.1. Patrones complejos	100
4.1.1. Maestro/Detalle	100
4.2. Componentes complejos	106
4.2.1. Ventana de selección.	107
4.2.2. Visor de Mapas	110
4.3. Tratamiento de ficheros	113
4.3.1. Manejo de ficheros de imágenes	113
4.3.2. Manejo de ficheros de cualquier tipo	114
4.3.3. Importar datos a la BD desde fichero	115
4.4. Control de la Navegación. Saltando entre ventanas.	116
4.4.1. Acumulador/Operador	117
4.4.2. Clave Ajena (salto modal)	120
4.5. Carga dinámica de clases	124
4.5.1. Introducción	124
4.5.2. Ejemplos de utilización	124
III. Complementos al desarrollo	125
5. Fuentes de datos	127
5.1. Conexiones BBDD	127
5.1.1. Bases de Datos accesibles por gvHidra	127
5.1.2. Acceso y conexión con la Base de Datos	127
5.1.3. Transacciones	133
5.1.4. Procedimientos almacenados	134
5.1.5. Recomendaciones en el uso de SQL	135
5.2. gvHidraDBAL	136
5.2.1. API de gvHidraDBAL	136
5.3. Web Services	148
5.3.1. Web Services en PHP	149
5.3.2. Web Services en gvHIDRA. WSComun	150
6. Seguridad	151
6.1. Autenticación de usuarios	151
6.1.1. Introducción	151
6.1.2. Crear un nuevo método de Autenticación	152
6.2. Modulos y Roles	155
6.2.1. Introducción	155
6.2.2. Uso en el framework	157
IV. Conceptos Avanzados	159
7. Conceptos Avanzados	161
7.1. Excepciones	161
7.1.1. gvHidraSQLException	161
7.1.2. gvHidraLockException	161
7.1.3. gvHidraPrepareException	161
7.1.4. gvHidraExecuteException	161
7.1.5. gvHidraFetchException	161
7.1.6. gvHidraNotInTransException	162
7.2. Log de Eventos	162
7.2.1. Introducción	162
7.2.2. Crear eventos en el log	163
7.2.3. Consulta del Log	163
7.3. Herramientas para debug de la aplicación.	164

7.3.1. xDebug	164
7.3.2. Consola del navegador (Firefox)	171
7.3.3. Debug de plantillas Smarty	173
7.4. CUSTOM	174
7.4.1. Pasos previos	174
7.4.2. Archivos CSS	174
7.4.3. Clases más usadas de bootstrap	175
7.4.4. Iconos mediante fuentes tipográficas (icon-fonts)	175
7.4.5. Tipos de clases	176
7.4.6. Cosas a tener en cuenta	176
7.4.7. Estilos generales de la aplicación	176
7.4.8. Personalización de Logos	177
7.4.9. Panel login	179
7.4.10. Pantalla de inicio	182
7.4.11. Acerca de	186
7.4.12. Ventanas de aviso	188
7.4.13. Partes genéricas de los paneles FIL LIS EDI	190
7.4.14. Partes genéricas de los paneles LIS EDI	196
7.4.15. Panel FIL	199
7.4.16. Panel EDI	201
7.4.17. Panel LIS	202
7.4.18. Panel maestro-detalle	205
7.4.19. Panel árbol	207
7.4.20. Solapas	208
7.4.21. Debugger	210
7.4.22. Ejemplos prácticos	211
V. Apendices	218
A. Pluggins, que son y como usarlos en mi aplicación.	221
A.1. Documentación Plugins gvHidra	221
A.1.1. cwarbol	222
A.1.2. cwareatexto	223
A.1.3. cwbarra	224
A.1.4. cwbarrainfpanel	225
A.1.5. cwbarrasuppanel	225
A.1.6. cwboton	226
A.1.7. cwbotontooltip	228
A.1.8. cwcampotexto	232
A.1.9. cwcontenedor	234
A.1.10. cwcheckbox	235
A.1.11. cwficha	236
A.1.12. cwfichaedicion	237
A.1.13. cwfila	237
A.1.14. cwimagen	238
A.1.15. cwinfocontenedor	239
A.1.16. cwinformation	240
A.1.17. cwlabel	241
A.1.18. cwlista	243
A.1.19. cwmarcopanel	245
A.1.20. cwmaps	245
A.1.21. cwmenulayer	247
A.1.22. cwpaginador	248
A.1.23. cwpanel	249
A.1.24. cwpantallaentrada	250
A.1.25. cwrichareatexto	251
A.1.26. cwslider	252
A.1.27. cwsolapa	253
A.1.28. cwtabla	254

A.1.29. cwupload	255
A.1.30. cwventana	256
B. Listados Jasper en gvHIDRA	259
B.1. Integración de Jasper en un mantenimiento	259
B.2. Depuración de errores al intentar ejecutar informes JasperReports en gvHidra	261
B.3. Recomendaciones y generalidades sobre integración gvHidra y Jasper	263
C. Pasos de migración para pasar a la versión 5.0.0	266
C.1. Codificación del proyecto.	266
C.2. Cambios en la lógica de negocio.	266
C.3. Migración de los ficheros XML de configuración.	267
C.4. Migración del fichero AppMainWindow.php	268
C.5. Migración del fichero mappings.php	268
C.6. Migración de las plantillas (TPL).	268
C.6.1. Plugins obsoletos.	268
C.6.2. Maestro/Detalle.	269
C.6.3. Migración del nombre de los plugins.	269
C.6.4. Notas importantes sobre los parámetros de los plugins.	270
C.6.5. Parámetros de plugins que se deben eliminar.	270
C.6.6. Parámetros de plugins que se deben renombrar.	271
C.6.7. Paso de la variable \$smt_y_iteracionActual al uso de la librería IgepJS.js	272
C.7. Migración de los views.	275
D. Novedades de la versión	276
D.1. Herencia en plantillas, plugins y ficheros de idiomas.	276
D.2. Mejora de rendimiento en la visualización de resultados.	277
D.3. Nuevos customs.	277
D.4. Menú de la aplicación.	278
D.5. Configuración aspectos generales de las ventanas.	280
D.5.1. Icono "modificado" en barra superior.	280
D.5.2. Ubicación botonera: Nueva búsqueda, listado, edición.	281
D.5.3. Redimensionar paneles.	282
D.5.4. Conteo de registros en las solapas de los detalles.	283
D.6. Configuración nuevas funcionalidades en paneles tabulares.	284
D.6.1. Opciones de selección de registros en un tabular.	284
D.6.2. Selección directamente en la fila.	285
D.6.3. Mostrar número total de registros en un tabular.	286
D.6.4. Alineación de campos en un panel tabular.	287
D.7. Nuevas operaciones y parámetros en plugins.	287
D.7.1. cwinfocontenedor. Operaciones getValue() y setValue().	287
D.7.2. cwinformation. Operaciones getValue() y setValue(). Parámetros nuevos.	288
D.7.3. cwbotontooltip con funcionamiento independiente del estado del panel.	289
D.7.4. Visibilidad de etiquetas que acompañan componentes.	289
D.7.5. Parámetro confirm en botones tooltip.	289
D.8. Debug en desarrollo.	290
D.8.1. Debug de plantillas en tiempo de ejecución en entorno de desarrollo.	290
D.8.2. Debug de JavaScript.	290

Lista de figuras

1.1. Ejemplo de patrón simple tabular.	4
1.2. Ejemplo de patrón simple registro.	4
1.3. Ejemplo de ventana de selección.	5
1.4. Ejemplo de lista desplegable con búsqueda incluida (select2).	5
1.5. Arquitectura MVC (Modelo-Vista-Controlador).	7
1.6. Estructura física MVC de un proyecto gvHidra.	8
1.7. Diagrama de secuencia de una llamada a gvHidra.	8
1.8. Resultado de ejecutar phpinfo().	10
1.9. Pantalla de inicio de sesión por defecto.	12
1.10. Pantalla de inicio con las opciones de menú que vienen por defecto.	13
2.1. Arquitectura de tres capas de gvHidra.	14
2.2. Anatomía de una ventana gvHidra.	15
2.3. Ej. Modo <i>búsqueda/tabla</i> (FIL/LIS).	16
2.4. Ej. Modo <i>búsqueda/ficha</i> (FIL/EDI).	17
2.5. Ej. Modo <i>búsqueda/tabla/ficha</i> (FIL/LIS/EDI).	17
2.6. Patrón <i>búsqueda</i> (FIL).	17
2.7. Patrón <i>tabular</i> (LIS).	18
2.8. Patrón <i>registro</i> (EDI).	19
2.9. Modo tabular (LIS).	19
2.10. Modo registro (EDI).	19
2.11. Ejemplo: Maestro registro (FIL/EDI) - Detalle tabular (LIS).	20
2.12. Ejemplo: Maestro tabular (FIL/LIS) - N Detalles.	21
2.13. Flujo en gvHidra ante un estímulo de intefaz de usuario.	22
3.1. Custom greyStyle	81
3.2. Custom lightStyle	81
3.3. Custom darkStyle	82
D.1. Custom greyStyle	277
D.2. Custom lightStyle	278
D.3. Custom darkStyle	278
D.4. Menú barra superior (custom darkStyle).	279
D.5. Menú barra lateral (custom darkStyle).	280
D.6. Panel modificado, icono barra superior.	280
D.7. Formato maximizado botonera (custom darkStyle).	281
D.8. Formato minimizado botonera (custom darkStyle).	282
D.9. Formato minimizado botonera con menú vertical (custom darkStyle).	282
D.10. Botón redimensionar.	283
D.11. Panel minimizado (estado por defecto).	283
D.12. Panel maximizado.	283
D.13. Maestro/nDetalles con conteo de registro en cada detalle.	284
D.14. Panel tabular con opciones de selección de registros.	285
D.15. Panel tabular con selección en fila activado.	286
D.16. Panel tabular con total de registros en cwtabla.	286
D.17. Panel tabular con total de registros en cwpaginador.	287
D.18. Alineación campos en panel tabular.	287
D.19. Plugin cwinfocontenedor.	288
D.20. Plugin cwinformation.	289
D.21. Mensaje de confirmación.	290

Lista de tablas

2.1. Conceptos comunes a todas las acciones genéricas.	22
2.2. Valores de retorno de los métodos virtuales.	22
3.1. Listado de Métodos 1	57
3.2. Listado de Métodos 2	57
4.1. Ficheros implicados en implementación	117
4.2. Ficheros implicados en implementación	120
5.1. Perfiles SGBD	128
5.2. Parámetros	140
5.3. Excepciones	140
5.4. Parámetros	140
5.5. Excepciones	140
5.6. Parámetros	140
5.7. Parámetros	141
5.8. Parámetros	141
5.9. Parámetros	141
5.10. Parámetros	142
5.11. Excepciones	142
5.12. Parámetros	142
5.13. Excepciones	142
5.14. Parámetros	142
5.15. Parámetros	143
5.16. Parámetros	143
5.17. Parámetros	143
5.18. Excepciones	144
5.19. Parámetros	144
5.20. Excepciones	144
5.21. Excepciones	144
5.22. Excepciones	144
5.23. Parámetros	145
5.24. Excepciones	145
5.25. Excepciones	145
5.26. Parámetros	145
5.27. Excepciones	145
5.28. Parámetros	146
5.29. Excepciones	146
5.30. Parámetros	146
5.31. Parámetros	146
5.32. Parámetros	146
5.33. Parámetros	147
5.34. Parámetros	147
5.35. Parámetros	147
5.36. Parámetros	147
5.37. Parámetros	147
5.38. Parámetros	148
5.39. Parámetros	148
5.40. Excepciones	148
6.1. Tabla de metodos 1	157
6.2. Tabla de metodos 2	157
7.1. Tabla de Excepciones	161
7.2. Tabla de clasificación de los eventos	162

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.	7
1.2.1. Hardware.	7
1.2.2. Software.	7
1.3. Entendiendo el entorno.	7
1.4. Instalación del entorno.	9
1.4.1. ¿El entorno funciona correctamente?	9
1.5. Creando mi “hola mundo” en gvHidra.	11
2. Conceptos técnicos de gvHidra.	14
2.1. Arquitectura	14
2.1.1. Arquitectura por capas.	14
2.1.2. Configuración.	15
2.2. Aspecto visual.	15
2.2.1. Anatomía de una ventana gvHidra.	15
2.2.2. Modos de trabajo y ventanas gvHidra.	16
2.2.3. Patrones de interfaz.	17
2.3. Lógica de negocio.	21
2.3.1. Acciones y operaciones.	21

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.

Pese a que la intención del documento es ser un manual de referencia más que un manual de introducción, a lo largo de los diferentes capítulos, se mostrarán las distintas posibilidades del *framework*, las mejoras o variaciones que aporta, frente a otras opciones y, por supuesto, cómo instalar y ejecutar una primera aplicación básica. Tras 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 de referencia, 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 existen otros medios a través de los cuales conocer y/o mejorar en el uso de gvHidra:

- Lista de distribución 'gvhidra_soporte' [http://listserv.gva.es/cgi-bin/mailman/listinfo/gvhidra_soporte]. Cualquier desarrollador puede suscribirse y hacer llegar sus peticiones de ayuda o sugerencias para mejorar la herramienta.

La lista de distribución está publica en la WEB (de forma similar a un foro) a través de 'nabble' [<http://gvhidra.3754916.n2.nabble.com>]

- El portal 'www.gvhidra.gva.es' [<http://www.gvhidra.gva.es>]. En el mismo se publica diferente información sobre la evolución del proyecto (tutoriales, seminarios y cursos, FAQs...)

1.1.2. ¿Qué es gvHIDRA?

gvHidra son las iniciales de "Generalitat Valenciana: **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 usabilidad y tal vez de aspecto 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 antigua Conselleria de Infraestructuras y Transporte de la Generalitat Valenciana (CIT) trabajaba 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 plantillas en PowerBuilder (tecnología de arquitectura cliente servidor) para reducir los tiempos de desarrollo a la vez que homogeneizaba los desarrollos favoreciendo el posterior mantenimiento.

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), cambiar a una arquitectura WEB en capas (minimizando así el trabajo de microinformática con instalaciones en un gran parque de PCs)...

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 productividad de nuestros

desarrollos y homogeneizar los mismos. Resolviendo además los nuevos requerimientos: el entorno web y el enfoque OpenSource.

Así, nace el 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 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 con las que surge el proyecto, pero ¿Se han logrado los objetivos? Enumerando alguna de las características del *framework* quizás quede constestada la pregunta:

- **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 comportamiento. Estos patrones definen la forma de representación de la información (formato tabular, registro,...) así como 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.

Figura 1.1. Ejemplo de patrón simple tabular.

Número Legislatura	Número Romano	Fecha Inicio	Fecha fin	Actual?
9	IX	11/06/2015		SI
8	VIII	09/06/2011	11/06/2015	NO
7	VII	14/06/2007	09/06/2011	NO
6	VI	12/06/2003	14/06/2007	NO
5	V	09/07/1999	12/06/2003	NO
4	IV	20/06/1995	09/07/1999	NO

Figura 1.2. Ejemplo de patrón simple registro.

Sra. []
 Nombre: Eva Apellido1: Alcón Apellido2: Soler
 NIF: 18935038N Iniciales: EAS
 Email: ealcom@corts.es
 URL: http://www.cortsvalecianas.es/cs/Satellite/Layout/Page/1,268674001642,FichaDiputados.html?param9=18935038N

- **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, etc. El *framework* genera estos componentes simplificando su utilización en las aplicaciones.

Figura 1.3. Ejemplo de ventana de selección.

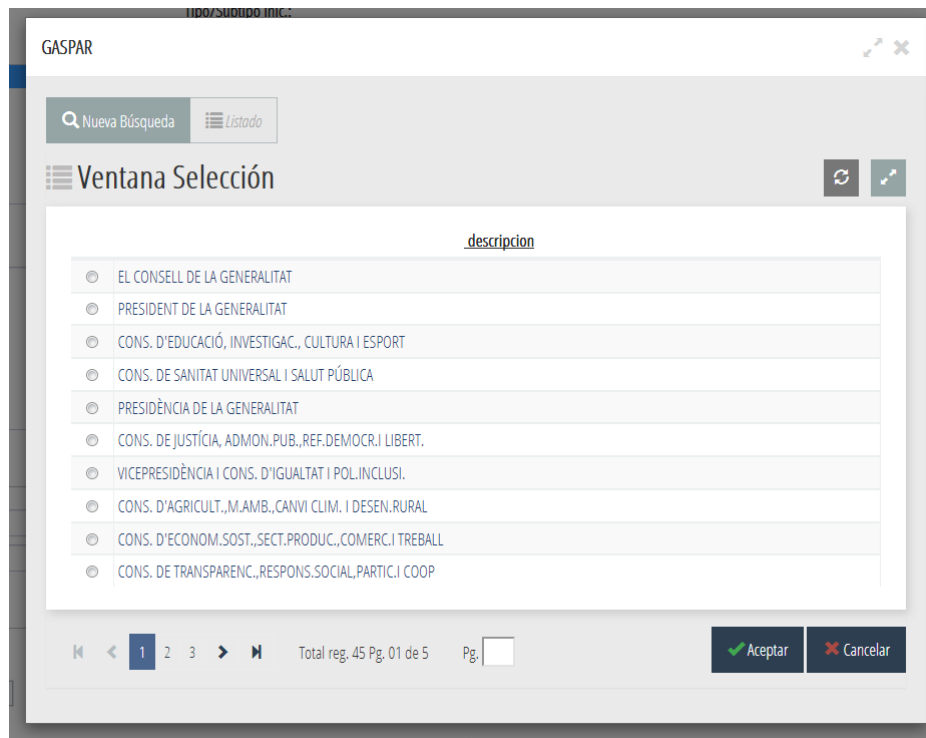
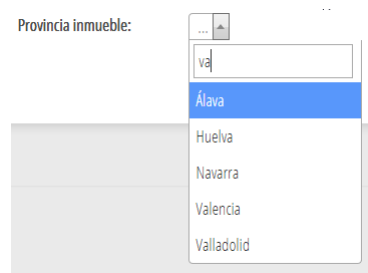


Figura 1.4. Ejemplo de lista desplegable con búsqueda incluida (select2).



- **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 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 PHP, el *framework* incorpora objetos persistentes y validación de tipos de datos.
- **Soporte a diferentes SGBD.**

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

- **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 objetivos principales de la herramienta es simplificar la labor del programador. Con esta premisa, hemos conseguido que un desarrollador de gvHIDRA realice una aplicación WEB en la mayoría de los casos 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 aspecto (CSS, imágenes, ...) como definir comportamientos propios de la organización.

- **Auditoría y depuración.**

Dispone de una herramineta de registro que sirve de base para auditorías y/o localizar errores en tiempo de ejecución y dar soporte concreto al usuario.

- **Facilidad de integración.**

- **Autenticación y autorización.**

Para poder integrarse con diferentes organismos incorpora un mecanismo de validación extensible siendo capaz de acoplarse a cualquier sistema de validación a través de PHP (LDAP, Active Directory, distintos SSO...)

- **Informes y reporting.**

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. Hay integración con *jasperreports* (proyecto JASPER), con librerías de generación en formato openDocument, etc.

- **Web services.**

Además de las facilidades nativas del propio PHP, gvHidra se apoya en el proyecto WSComun para consumir servicios basados en WSSecurity.

Como proyecto *opensource*, gvHidra sigue 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 5.0.0 (rama 5.0.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, y 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 explican las necesidades de un puesto cliente (desarrollador) para poder ejecutar una aplicación realizada con gvHidra.

1.2.1. Hardware.

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

- Pentium IV 2GHz o superior
- RAM 512 MB o superior

1.2.2. Software.

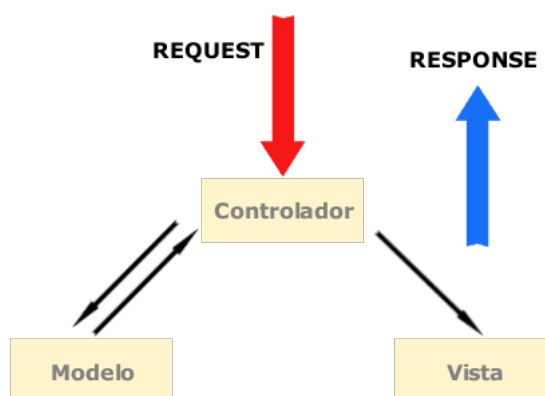
En el caso de los requerimientos de software **mínimos**:

- Firefox ESR 45.9.0 o superior.
- Google Chrome 52 o superior.
- Visor de PDFs (recomendado Acrobat Reader con soporte de firma digital o similar).

1.3. Entendiendo el entorno.

El *framework* gvHIDRA sigue una arquitectura MVC (Modelo-Vista-Controlador). 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:

Figura 1.5. Arquitectura MVC (Modelo-Vista-Controlador).



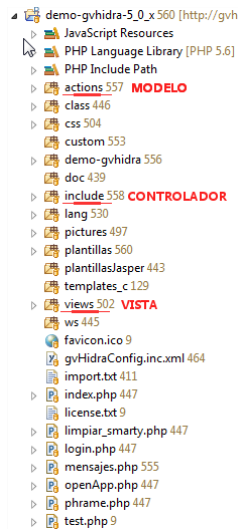
Esta división en capas se plasma físicamente dentro de una aplicación gvHIDRA. Concretamente, para hacer un mantenimiento, el programador tendrá que trabajar en las siguientes partes de la estructura:

- **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 Ccontrolador. Basado en el proyecto Phrame (a su vez inspirado en Apache Struts), 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.

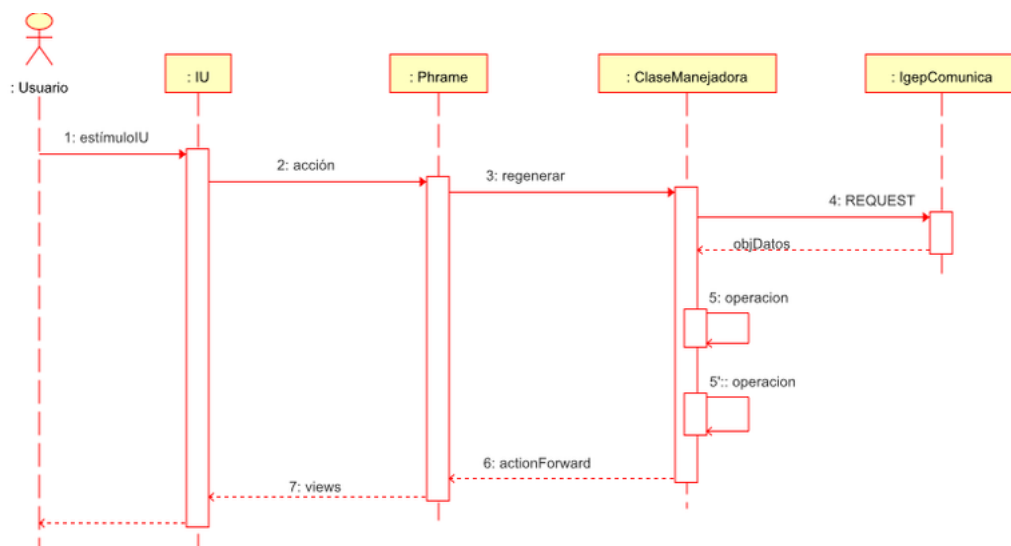
Figura 1.6. Estructura física MVC de un proyecto gvHidra.



A estas capas, falta añadir la **interfaz de usuario**, es decir, las pantallas. Para la realización de las mismas, gvHIDRA hace uso del motor de plantillas *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.

Figura 1.7. Diagrama de secuencia de una llamada a gvHidra.



1. El flujo se inicia con la aparición de un estímulo de pantalla lanzado por el usuario.
2. Este estímulo es transmitido al controlador (en nuestro caso *Phrame*) en forma de acción. Ahora es *Phrame* quien, consultado con el fichero de mapeos (fichero *mappings.php*) es capaz de saber qué clase es la encargada de ges-

tionar la acción. Una vez conocida la clase, la inicia (ya sea creándola, o ya sea restaurándola de la sesión si ya había sido cargada previamente) 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 una redirección (*actionForward*) a *Phrame*. Éste último lo descompone y se localiza la vista seleccionada.
5. Finalmente, la vista (*views*) recoge la información y, con la plantilla (archivo *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 interpretar PHP. Estos son los requisitos técnicos:

- **Servidor WEB.** Puede ser cualquier servidor, aunque nosotros recomendamos Apache 2.X.
- **PHP.** Las versiones 4.2, 4.4 y 5.0 de gvHidra son compatibles con PHP 5.4 y 5.6
- **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 (ó SGBDs).
 - **PEAR:** sistema básico de PEAR

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 raíz del servidor web (generalmente, htdocs o www) 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.

Figura 1.8. Resultado de ejecutar phpinfo().

System	Linux dsahidra01.srv.gva.es 2.6.32-696.23.1.el6.x86_64 #1 SMP Wed Mar 14 02:11:19 EDT 2018 x86_64
Build Date	Jul 30 2018 12:43:21
Configure Command	'/configure' '--prefix=/srv_apl/gvhidra/httpd-2.4.33_php-5.6.36' '--with-oci8=shared,installclient,/srv_apl/gvhidra/oracle/installclient_11_2' '--with-libdir=lib64' '--enable-mbstring' '--with-gd' '--with-zlib' '--with-freetype-dir=/usr/lib64' '--enable-soap' '--with-apxs2=/srv_apl/gvhidra/httpd-2.4.33_php-5.6.36/bin/apxs' '--with-pear' '--with-curl' '--with-pgsql=/srv_apl/gvhidra/postgresql/postgresql-9.3.2' '--with-pdo-pgsql=/srv_apl/gvhidra/postgresql/postgresql-9.3.2' '--with-pdo-oci=installclient,/srv_apl/gvhidra/oracle/installclient_11_2,11.2' '--with-libxml-dir=/usr/include/libxml2' '--with-png-dir=/usr' '--with-jpeg-dir=/usr' '--enable-zip' '--with-openssl=/srv_apl/openssl'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/srv_apl/gvhidra/httpd-2.4.33_php-5.6.36/lib
Loaded Configuration File	/srv_apl/gvhidra/httpd-2.4.33_php-5.6.36/lib/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,NTS
PHP Extension Build	API20131226,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, mcrypt.*, mdcrypt.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.6.0. Copyright (c) 1999-2016 Zend Technologies

Podemos aprovechar para comprobar que la versión de PHP es la correcta, que están activas las librerías de acceso a base de datos, así como cualquier otra librería que requiera la aplicación.

1.4.1.2. ¿Funciona la conexión al SGBD?

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 mediante 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' );

$dns = array (
    'phptype' => 'xxxx' ,
    'username' => 'xxxx' ,
    'password' => 'xxxx' ,
    'hostspec' => 'xxxx' ,
    'port' => 'xxxx'
);

$options = array ( 'portability' => MDB2_PORTABILITY_NONE );
$res = MDB2::connect ( $dns, $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 situar 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.gva.es>] y descargamos la última versión de gvHidra.

En el paquete ZIP descargado, tendremos los siguientes componentes:

- **Directorio appTemplate**: Aquí encontramos la estructura básica de directorios y ficheros para comenzar a desarrollar una aplicación.
- **Directorio custom**: En él encontraremos los diferentes temas que se pueden utilizar a la hora de desarrollar una aplicación gvHIDRA.
- **Directorio genaro**: Herramienta para generar código gvHIDRA.
- **Directorio igep**: Contiene el core del framework. Es el directorio que deberemos copiar en todas nuestras aplicaciones para poder trabajar con gvHIDRA.
- **Directorio samples**: En él se incluirán ejemplos de uso de determinadas funcionalidades. Por ejemplo, usos de formas de validación.
- **guia_rapida.txt**: Pasos mínimos de prueba del paquete.

- **gvHIDRA_MANUAL.pdf**: Manual completo del framework.
- **license.txt**: Referencia a la licencia GPL.
- **readme.txt**: Información básica del proyecto.

Una vez descargado, seguiremos los siguientes pasos:

1. Copiamos el directorio **appTemplate** en una carpeta del htdocs del servidor web.

Si renombramos la carpeta por un nombre más identificable para la aplicación, debemos dar el mismo nombre a una carpeta que se encuentra dentro también con el nombre *appTemplate*

2. Lo siguiente es copiar la carpeta *igep* que se encuentra en el paquete descargado dentro de la carpeta creada en el paso anterior.

Nota: puedes comprobar que la estructura es similar a la que hemos presentado en la imagen del punto 3.

3. Al directorio *templates_c* (se debe crear si no existe) le otorgamos permiso de escritura para el usuario Apache. Este directorio es el que utiliza el *framework* para compilar las plantillas.
4. Acceder a <http://<servidor>/appTemplate> 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:

Figura 1.9. Pantalla de inicio de sesión por defecto.



Tras validarnos como usuario, entramos en la ventana principal de la aplicación que nos muestra las opciones de menú.

Figura 1.10. Pantalla de inicio con las opciones de menú que vienen por defecto.



Enhorabuena ¡Ya tienes la base de tu primera aplicación! Ahora puedes hacer uso del generador de código de gvHidra, *Genaro*, para crear las ventanas de mantenimiento.

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.

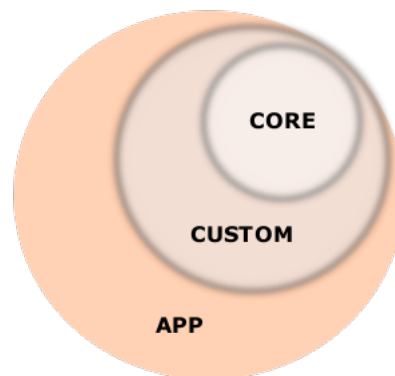
Se verán conceptos relativos a la arquitectura de las aplicaciones, a la interfaz con el usuario y a la lógica de negocio.

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 configuración 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.

Figura 2.1. Arquitectura de tres capas de gvHidra.



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

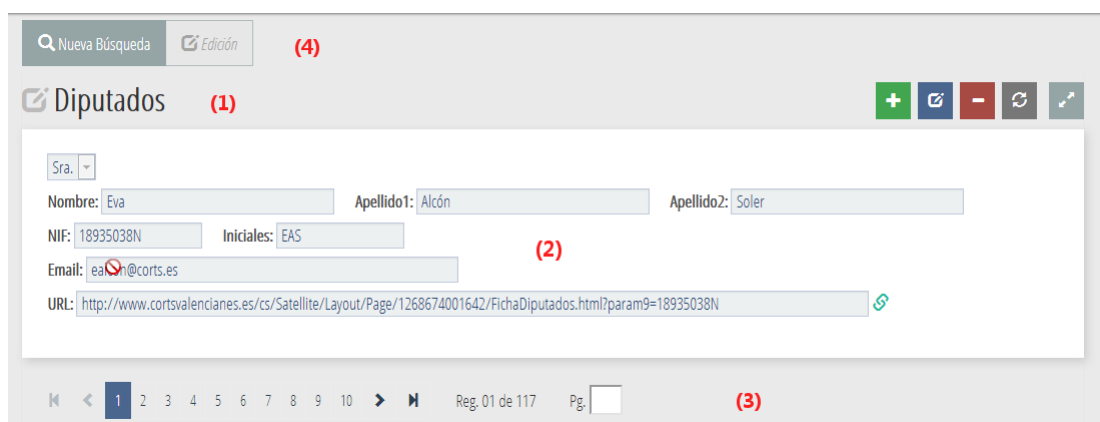
- **carga estática:** esta fase carga los parámetros que estén ubicados en el fichero de configuración *gvHidraConfig.inc.xml*. Tenemos un fichero de este tipo en cada una de las capas, se cargarán en este orden:
 - *igep/gvHidraConfig.inc.xml*: fichero de configuración propio del framework. No debe 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:
 - *igep/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.

Figura 2.2. 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. Los *Botones tooltip* son botones que efectúan acciones relacionadas con la interfaz.

- **Contenedor (2)**

Zona 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)*, de modo que la pestaña resaltada 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 una ficha, respectivamente), donde los datos ya son accesibles y se podrá operar sobre ellos.

Figura 2.3. Ej. Modo búsqueda/tabla (FIL/LIS).

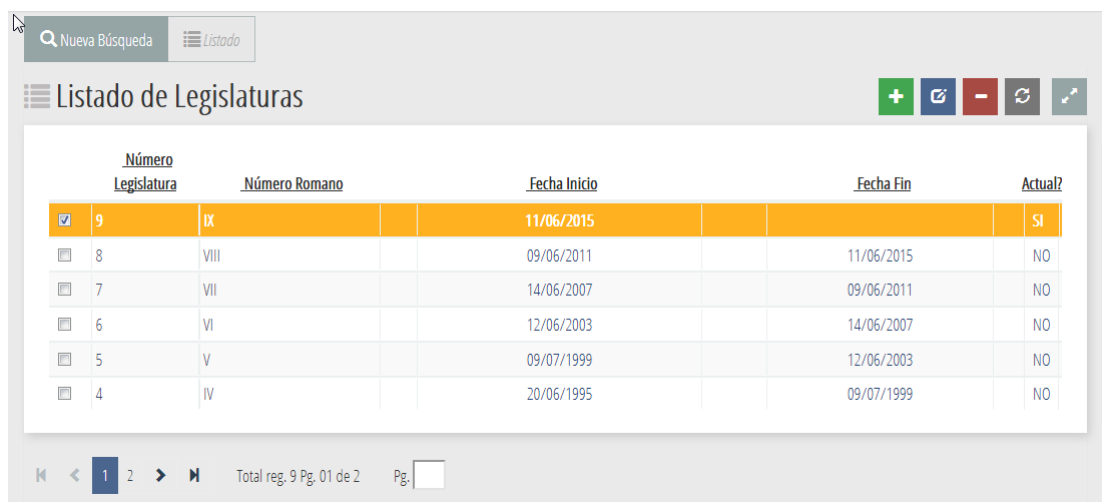
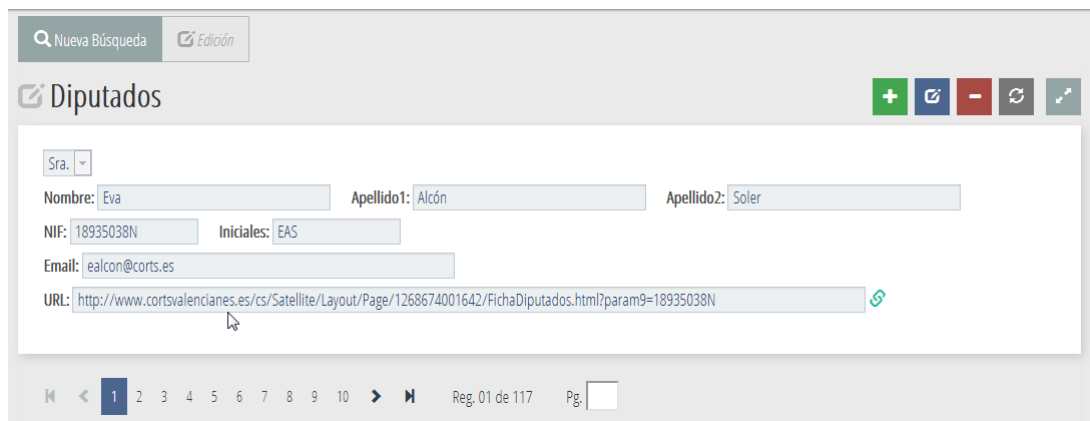


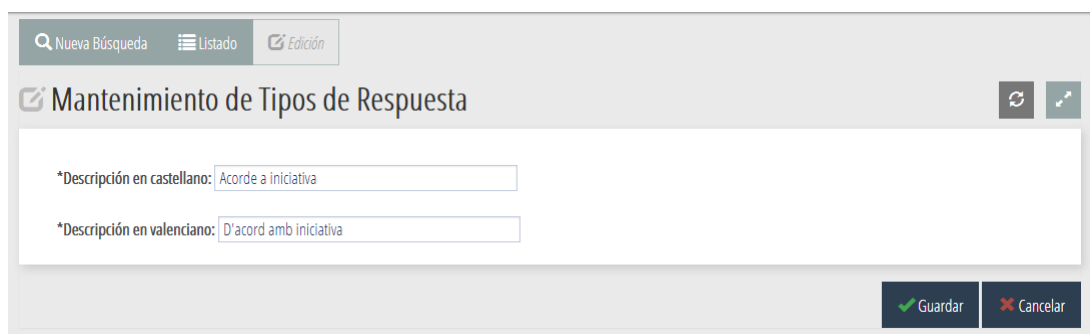
Figura 2.4. Ej. Modo búsqueda/ficha (FIL/EDI).



2. Tres modos de trabajo.

En este caso es una combinación de los dos anteriores. Partiendo 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 desde el panel tabular, mientras que para inserción o modificación se pasará a un panel modo ficha.

Figura 2.5. Ej. Modo búsqueda/tabla/ficha (FIL/LIS/EDI).

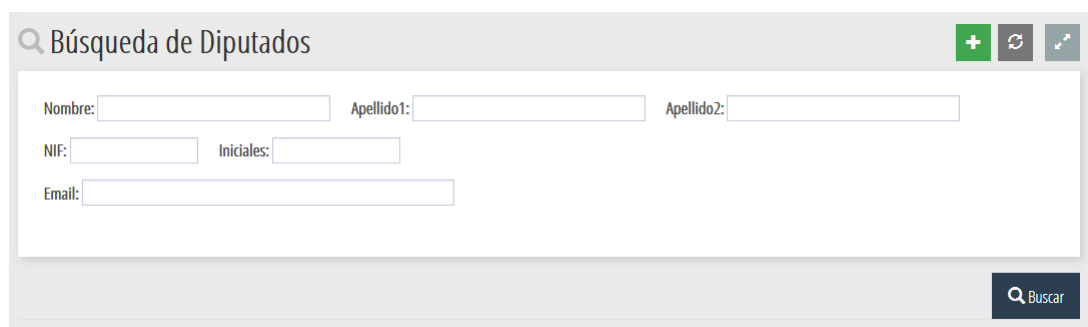


2.2.3. Patrones de interfaz.

Una vez vista 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:

Figura 2.6. Patrón búsqueda (FIL).



[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. [18]
2. Registro. [18]
3. Tabular-Registro. [19]
4. Maestro-detalle. [19]

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.

Figura 2.7. Patrón tabular (LIS).

	Número Legislatura	Número Romano	Fecha Inicio	Fecha Fin	Actual?
<input checked="" type="checkbox"/>	9	IX	11/06/2015		SI
<input type="checkbox"/>	8	VIII	09/06/2011	11/06/2015	NO
<input type="checkbox"/>	7	VII	14/06/2007	09/06/2011	NO
<input type="checkbox"/>	6	VI	12/06/2003	14/06/2007	NO
<input type="checkbox"/>	5	V	09/07/1999	12/06/2003	NO
<input type="checkbox"/>	4	IV	20/06/1995	09/07/1999	NO

[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.

Figura 2.8. Patrón registro (EDI).

[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.

Figura 2.9. Modo tabular (LIS).

	Descripción en castellano	Descripción en valenciano	F. Baja
<input type="checkbox"/>	Acorde a iniciativa	D'acord amb iniciativa	
<input type="checkbox"/>	Puesta a disposición	Posada a disposició...	
<input type="checkbox"/>	Escrita	Escrita	

Figura 2.10. Modo registro (EDI).

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.

Figura 2.11. Ejemplo: Maestro registro (FIL/EDI) - Detalle tabular (LIS).

The screenshot shows a web application interface with two main sections. The top section, titled 'Diputados', contains a form with the following fields: 'Sra.' (dropdown), 'Nombre: Eva', 'Apellido1: Alcón', 'Apellido2: Soler', 'NIF: 18935038N', 'Iniciales: EAS', 'Email: ealcon@corts.es', and 'URL: http://www.cortsvalencianes.es/cs/Satellite/Layout/Page/1268674001642/FichaDiputados.html?param9=18935038N'. Below the form is a pagination control showing 'Reg. 01 de 117' and 'Pg. 1'. The bottom section, titled 'Grupos', contains a table with the following data:

Legislatura	Grupo	Circunscripción	F. Ini.	F. Fin.	Motivo Baja
IX	Socialista	Castellón	11/06/2015	26/04/2017	Vuelta a la actividad docente e investigadora. Sustituida por Ana

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.

Figura 2.12. Ejemplo: Maestro tabular (FIL/LIS) - N Detalles.

The screenshot displays two data tables in a web interface. The top table, 'Listado de Legislaturas', has columns: 'Número Legislatura', 'Número Romano', 'Fecha Inicio', 'Fecha Fin', and 'Actual?'. It lists periods from IV to IX, with period IX highlighted in orange. The bottom table, 'Listado órganos', has columns: 'Abrev. Cas.', 'Abrev. Val.', and 'Fecha Fin'. It shows a single entry 'Pendiente asignación'.

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 interfaz de usuario). Las acciones están divididas en dos grupos:

1. Acciones genéricas:

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

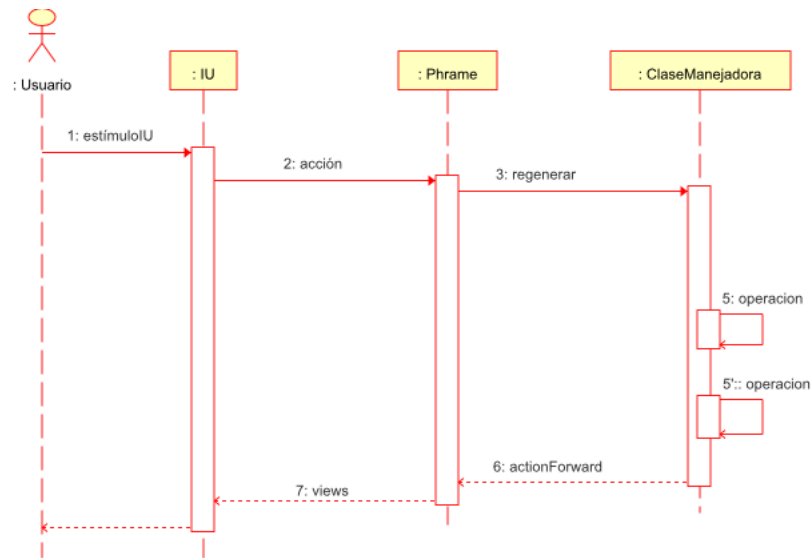
2. Acciones particulares:

Con las acciones particulares se podrán resolver necesidades concretas no resueltas automáticamente 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 por defecto de la operación sobrescribiendo 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.

Figura 2.13. Flujo en gvHidra ante un estímulo de interfaz de usuario.



Vamos a explicar un poco más en profundidad cada una de las acciones genéricas, detallando tanto el objetivo global de la acción como 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.

<p>Métodos virtuales: Todas las acciones genéricas tienen una serie de métodos que el programador puede sobrescribir 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.</p> <p>Ejemplos:</p> <p><i>preInsertar, postInsertar, preModificar, postModificar, ...</i></p> <p>actionForward: Acción de retorno de un método virtual. Puede tomar un valor por defecto asignado por el framework o el programador asignarle uno particular.</p> <p>Ejemplos:</p> <p><i>gvHidraSuccess, gvHidraSuccessOne, gvHidraError, gvHidraNoData, gvHidraNoAction, ...</i></p>
--

Tabla 2.2. Valores de retorno de los métodos virtuales.

<p>Valor 0: Resultado correcto, la ejecución continua.</p> <p>En este caso el <i>actionForward</i> se corresponde con la acción gvHidraSuccess.</p> <p>Valor -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: <i>\$this->showMessage('xxx');</i>) No se recargará la ventana.</p> <p>En este caso el <i>actionForward</i> se corresponde con la acción gvHidraError.</p> <p>actionForward: Acción de retorno programada por el desarrollador para saltarse la ejecución de por defecto, provoca una recarga de la ventana.</p> <p>Común a todas las acciones genéricas salvo a la de <i>Recargar</i>, esta acción no contempla una acción de retorno particular.</p>

2.3.1.1. Acciones genéricas.

1. Iniciar ventana.

La acción genérica **Iniciar ventana** se ejecutará al iniciar la ventana, y 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.

La acción genérica **Buscar** realiza la consulta del panel basándose en los datos que hay en el panel filtro. La acción *buscar* se compone de dos operaciones:

- **buildQuery**: Método que recoge los datos del panel filtro y los añade al WHERE de la consulta del panel. Su comportamiento se puede sobrecargar con 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. También es importante saber 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 (LIS-EDI), existe la posibilidad de añadir el *actionForward* con la acción *gvHidraSuccessOne* que permite agilizar el uso de la ventana cuando la búsqueda devuelve un único registro pasando directamente al modo edición. Se ejecutarán los mismos métodos que al editar un registro.

3. Editar.

La acción genérica **Editar** se ejecuta cuando estamos trabajando con tres modos (FIL/LIS/EDI). 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 que recoge los datos del panel registro (EDI) 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 que se encarga de realizar la consulta que se ha generado con el método anterior, obteniendo así los datos actualizados. También es importante saber 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

La acción genérica **Recargar** actúa en un patrón maestro-detalle. Se ejecutará cuando se efectúe un cambio de maestro, y recargará los detalles correspondientes. La acción se compone de dos operaciones:

- **buildQueryDetails:** 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 que se encarga de realizar la consulta que se ha generado en el método anterior, obteniendo 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

La acción genérica **Modificar** 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 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 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 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 método *postBuscar* o el método *postRecargar*.

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

6. Borrar

La acción genérica **Borrar** se dispara “típicamente” desde el modo de trabajo tabular para eliminar las tuplas seleccionadas (aunque es posible lanzarse tanto desde el modo búsqueda como desde el modo registro). 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 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 método *postBuscar* o el método *postRecargar*.

7. Insertar

La acción genérica **Insertar** 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 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

La acción genérica **Nuevo** se invoca para preparar la presentación de datos en pantalla antes de una inserción (nótese que es una acción previa y distinta a *Insertar*, siendo *Insertar* la propia acción de insertar en la base de datos, mientras que *Nuevo* es la acción previa a mostrar el formulario en modo inserción para que el usuario rellene los campos deseados). La acción se compone de la siguiente operación:

- **nuevo**: Realizará las operaciones de preparación de los datos y visualización previos a la edición para inserción. Esta operación se puede sobrescribir 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 que es calculado a partir de los valores de otros campos.

Validaciones de las acciones genéricas: las acciones genéricas recién enumeradas 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*. En esos casos, deshabilitar esta validación permitirá 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.	28
3.1. Estructura de aplicación.	28
3.1.1. Configuración estática: <i>gvHidraConfig.xml</i>	28
3.1.2. Configuración dinámica: <i>AppMainWindow</i>	33
3.1.3. Recomendaciones.	34
3.2. Breve guía para crear una pantalla	37
3.2.1. Introducción	37
3.2.2. Genaro: Generación automática en gvHidra	37
3.2.3. Revisión de un mantenimiento	42
3.3. Menú de una aplicación	49
3.3.1. Funcionamiento del menú	49
3.3.2. Opciones predefinidas para el menú	54
3.3.3. Autenticación y autorización (módulos y roles)	55
3.4. Diseño de pantalla con Smarty/plugins	58
3.4.1. ¿Qué es un template?	58
3.4.2. Cómo realizar una plantilla (tpl) básica.	58
3.4.3. Diseño avanzado de plantillas.	61
3.4.4. Documentación de los plugins	62
3.5. Código de la lógica de negocio	62
3.5.1. Operaciones y métodos virtuales	62
3.5.2. Uso del panel de búsqueda	72
3.5.3. Acciones no genéricas	75
3.5.4. Acciones de interfaz	78
3.6. Personalizando el estilo	81
3.7. Tratamiento de tipos de datos.	82
3.7.1. Características generales	82
3.7.2. Cadenas (<i>gvHidraString</i>)	83
3.7.3. Fechas	84
3.7.4. Números	88
3.7.5. Creación de nuevos tipos de datos	89
3.8. Listas de datos	90
3.8.1. Listas	90
3.8.2. Checkbox.	95
3.9. Mensajes y Errores.	96
3.9.1. Invocación desde código.	98
3.9.2. Invocación como confirmación.	98
3.10. Uso de datos por defecto.	99
4. Elementos de pantalla avanzados	100
4.1. Patrones complejos	100
4.1.1. Maestro/Detalle	100
4.2. Componentes complejos	106
4.2.1. Ventana de selección.	107
4.2.2. Visor de Mapas	110
4.3. Tratamiento de ficheros	113
4.3.1. Manejo de ficheros de imágenes	113
4.3.2. Manejo de ficheros de cualquier tipo	114
4.3.3. Importar datos a la BD desde fichero	115
4.4. Control de la Navegación. Saltando entre ventanas.	116
4.4.1. Acumulador/Operador	117
4.4.2. Clave Ajena (salto modal)	120
4.5. Carga dinámica de clases	124
4.5.1. Introducción	124
4.5.2. Ejemplos de utilización	124

Capítulo 3. Elementos básicos.

3.1. Estructura de aplicación.

El framework **gvHIDRA**, además de proporcionar un núcleo (*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 directorio `/doc/appTemplate` dentro del paquete de la propia versión. Antes de descomprimir hay que advertir que los ficheros están **codificados en UTF8** en la versión 5 de gvHIDRA (mientras que en anteriores versiones estaban en ISO-8859-1), por lo que hay que tener precaución con usar algún editor que respete estas codificaciones.

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

- **gvHidraConfig.inc.xml**: donde se indicará la versión de la aplicación y la ruta al fichero de configuración de la aplicación.
- **externo.gvHidraConfig.inc.xml**: donde añadir las conexiones a bases de datos, código de la aplicación, custom, log...
- **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 menú 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 sobre éste directorio.

A esta estructura tenemos que añadirle el núcleo 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 (corresponderá al fichero `gvHidraConfig.xml` que estará 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 "*tecni-map*" los ficheros de configuración se ubicarán en la estructura de directorios como sigue:

- *factur/igep/gvHidraConfig.inc.xml*
- *factur/custom/tecni-map/gvHidraConfig.inc.xml*
- *factur/gvHidraConfig.inc.xml*

La semántica de los ficheros XML vendrá determinada a través de la DTD que se encuentra en el paquete "*igep*" en el directorio **igep/dtd**. En el directorio se encuentran dos DTD:

- **configAPP.dtd**

DTD que corresponderá al fichero de configuración a nivel de aplicación (ej. *factur/gvHidraConfig.inc.xml* y *factur/factur/externo.gvHidraConfig.inc.xml*).

- **configFW.dtd**

DTD que corresponderá al fichero de configuración del framework.

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.
- **Nivel de organización** (personalización o custom, en el ejemplo anterior es el fichero *factur/custom/tecni-map/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, facilitar tareas de configuración a los equipos de sistemas, incrementar la seguridad valores de configuración fuera del *document_root*, etc. (ver más adelante propiedad *ext-ConfigDir*)

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 conviene que las conozcamos:

- **applicationName**: De carácter obligatorio, establece el nombre (código identificador) 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.

- **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 el 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.

- **customTitle:** Breve descripción de la aplicación. Es decir, el 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.
- **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.
- **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.
- **reloadMappings:** Indica si se quiere recargar el fichero de mappings en cada petición. Por defecto está habilitado, aunque habría 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.
- **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 "lightStyle". 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.

La ubicación del directorio custom se puede, permitiendo que evolucione 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

- **logSettings:** Establece el nivel de detalle de los registros de auditoría.
 - **LOG_ALL:** Mostrará todos los errores
 - **LOG_NONE:** No mostrará los errores. En producción es aconsejable este valor para no ralentizar la ejecución de la aplicación.
- **logJSSettings:** Establece el nivel de detalle de los registros de auditoría correspondiente a la parte de interfaz (JavaScript).
 - **LOG_ALL:** Mostrará todos los errores
 - **LOG_NONE:** No mostrará los errores. En producción es aconsejable este valor para no ralentizar la ejecución de la aplicación.
- **breadcrumb:** Mostrará la ubicación del módulo en el que nos encontramos en la pantalla principal.
- **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 conexiones 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 requiere trazabilidad, se indica el identificador de la aplicación que realiza la invocación. El atributo opcional *enablePAITrace* 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.
- **langZone:** Sección que contiene los diferentes idiomas de la aplicación.
- **lang:** Sección que contiene los diferentes idiomas de la aplicación.

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>
<barTitle>FACTUR</appVersion>
<templatesCompilationDir>FACTUR</templatesCompilationDir>
<customDirName path="custom">lightStyle</customDirName>
<reloadMappings>true</reloadMappings>
<smartyCompileCheck>true</smartyCompileCheck>
<logSettings status='LOG_ALL' dsnRef='g_dsn_log'>
<logJsSettings status='LOG_ALL' dsnRef='g_dsn_log'>
<queryMode status='2'>

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

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

  <dbDSN id='ora-thin' sgbid='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 o 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.
- **logJSSettings:** dependiendo de la configuración del entorno es mejor activar o desactivar esta configuración. En desarrollo es mejor activarlo (LOG_ALL), y desactivarlo al publicar en producción (LOG_NONE) para mejorar el rendimiento.

3.1.2. Configuración dinámica: AppMainWindow.

La clase **AppMainWindow** es propia de cada proyecto, y se crea para inicializar características 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 *gvHydraConfig.inc.xml*. Sin embargo, cuando necesitamos fijar alguno de estos parámetros en función 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 aplicación, la definición de esta clase nos permite acceder a estas acciones específicas. Vease el ejemplo en la Sección 3.5, “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', 'gvHydraOpenApp', 'index.php?view=igep/views/aplicacion.php');
$this->_AddForward ('abrirAplicacion', 'gvHydraCloseApp', 'index.php?view=igep/views/gvHydraCloseApp.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 selección (**gvHidraSeleccionWindow**) a nuestras aplicaciones necesitamos definirlos en el constructor de esta clase. Hay más información al respecto en la Sección 3.8, “Listas de datos” y Sección 4.2.1, “Ventana de selección.”, respectivamente.

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, sí 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 **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 *gvHidraConfig.inc.xml* a nivel de aplicación:

- **reloadMappings**: esta propiedad que admite valores booleanos(true|false), indica si el framework debe recargar el mapeo de acciones en cada iteración. Esto es conveniente en desarrollo, aunque ralentiza la ejecución. *En explotación esta propiedad debe estar a false.*
- **smartyCompileCheck**: esta propiedad que admite valores booleanos(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 **UTF8**, por lo que el IDE que utilicemos para programar, debemos configurarlo en esta codificación. Con ello evitaremos problemas con caracteres especiales.

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 Rápida

Aunque no es obligatorio, siempre conviene tener una guía para orientar 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 de usuario" descripcion="Manual de uso de la aplicación"
url="phrame.php?action=OpenManual__open" abrirVentana='true' iconCSS="glyphicon
glyphicon-book" />
```

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 siguiente opción de menú:

```
<opcion titulo="Acerca de..." url="about" iconCSS="glyphicon glyphicon-info-sign" />
```

Al pulsar sobre esta opción el usuario 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.

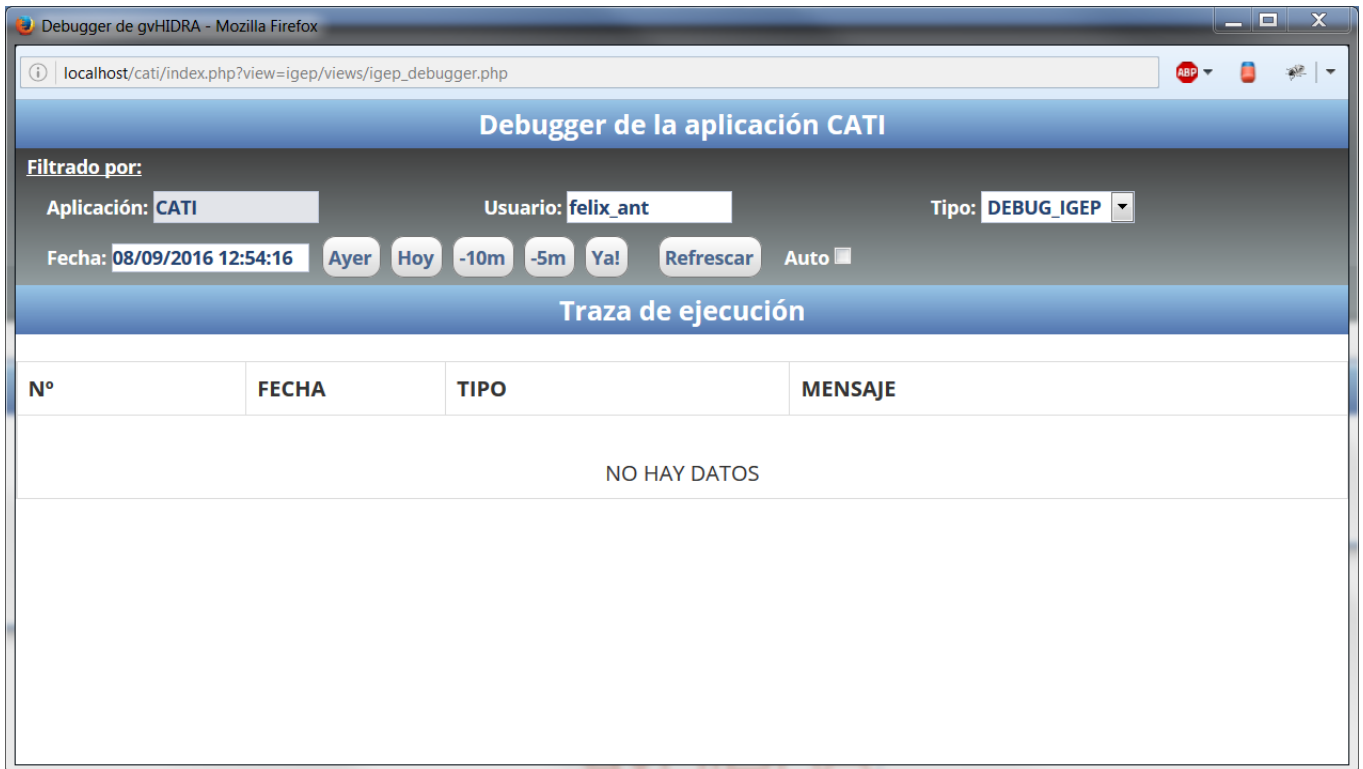
3.1.3.2. Programando...

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" iconCSS="glyphicon glyphicon-bookmark">
<controlAcceso>
...
</opcion>
```

</controlAcceso>
</opcion>

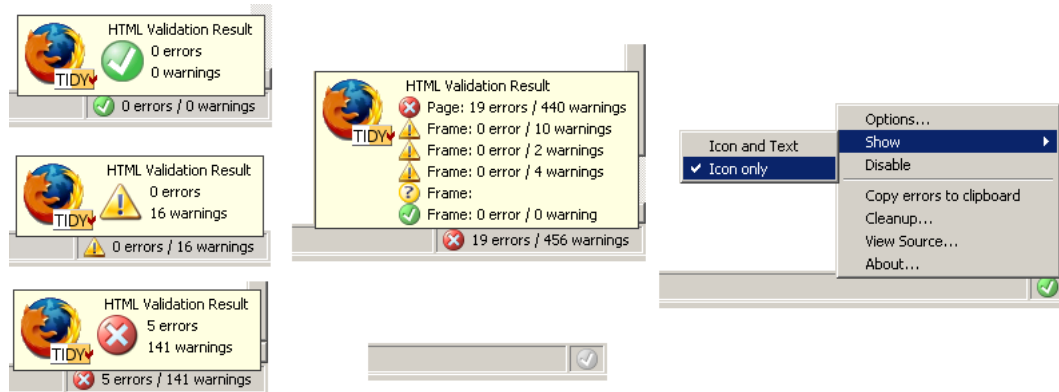


Más adelante explicaremos en profundidad el log del framework y sus posibilidades a modo de depuración y auditoría.

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.

3.2. 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 interactuá 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: genera 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.

- Genaro 5.0.0 [xxx]

Una vez descargado, lo dejamos en el directorio include de nuestra aplicación gvHidra. Si no tenemos aún 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 raíz 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 SÓLO 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 generar un patrón simple 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.
- Clase Manejadora: el nombre de la clase.
- Tabla de la bd: tabla sobre la que queremos realizar el mantenimiento.
- Patrón: seleccionamos la forma de visualización 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.
- Sección 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.
- Sección 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

GENARO v.5.0.0
Generador de Código gvHIDRA

Patrón Simple | **Patrón Maestro-Detalle** | Eliminar Clase | Conexión g_dsn

Las claves primarias del Maestro y claves Foráneas de los Detalles deben separarse por comas

Nombre Módulo:

Maestro

Clase Manejadora: Seleccionar Tabla: ⚠

Clave Primaria: Seleccionar Patrón:

Numero detalles:

Detalle 1

Clase Manejadora: Seleccionar Tabla:

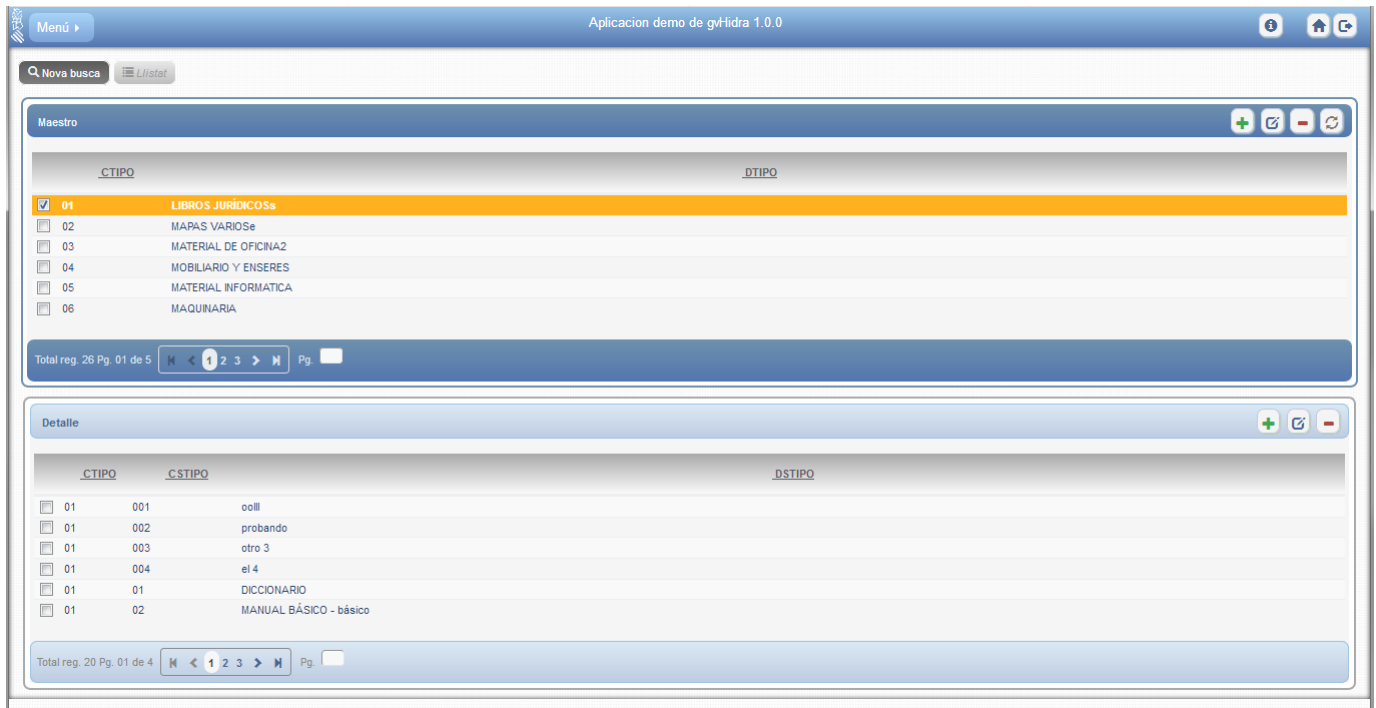
Clave Ajena: Seleccionar Patrón:

✓ **Maestro-Detalle P2M2FIL-LISM1LIS generado con éxito en el módulo MaestroDetalle**

ℹ **Recuerde que debe volver a entrar en la aplicación para ver la nueva ventana generada.**

FONDOS ESTRUCTURALES | Unión Europea

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 Tinventados 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 - Correspondencias 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 [127].
- 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 **setSelectForEditQuery()**. 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 [100].

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 [21].

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');
?>
```

- 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 [58].

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 caracteres especiales como acentos o eñes conviene ponerlos usando entidades HTML [<http://blog.fidelizador.com/2017/09/27/utf8-n-tildes-y-caracteres-especiales/>], 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 (
) 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 ...}&nbsp;{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 layout="botonera-classic" tipoAviso=$smarty_tipoAviso codAviso=
$smarty_codError descBreve = $smarty_descBreve textoAviso=$smarty_textoAviso onLoad=
$smarty_jsOnLoad}
```

```
{cwbarra usuario=$smtty_usuario codigo=$smtty_codigo customTitle=$smtty_customTitle
modal=$smtty_modal iconOut="glyphicon glyphicon-log-out" iconHome="glyphicon
glyphicon-home" iconInfo="glyphicon glyphicon-info-sign"}
{cwmenulayer name="$smtty_nombre" arrayMenu=$smtty_arrayMenu}
{/cwbarra}
{cwmarcopanel}

{cwpanel id="fil" action="buscar" method="post" estado="$estado_fil"
claseManejadora="TinvEstados"}
  {cwbarrasuppanel titulo="Estados de los bienes"}
    {cwbotontooltip titulo="Limpiar campos" accion="limpiar" actuaSobre="ficha"}
  {/cwbarrasuppanel}
  {cwcontenedor}
    {cwficha}
      <br />
      {cwcampotexto nombre="filCodEstado" size="3" editable="true"
label="Código" value=$defaultData_TinvEstados.filCodEstado dataType=
$dataType_TinvEstados.filCodEstado}
      <br /><br />
      {cwcampotexto nombre="filDescEstado" size="10" editable="true"
label="Descripción" value=$defaultData_TinvEstados.filCodigoEstado dataType=
$dataType_TinvEstados.filCodigoEstado}
      <br /><br />
    {/cwficha}
  {/cwcontenedor}
  {cwbarrainfpanel}
    {cwboton id="BtnBuscarFil" iconCSS="glyphicon glyphicon-search" label="Buscar"
class="button" accion="buscar" mostrarEspera="true"}
  {/cwbarrainfpanel}
{/cwpanel}

<!-- ***** MODO lis *****-->
{cwpanel id="lis" tipoComprobacion="envio" action="operarBD" method="post"
estado="$estado_lis" claseManejadora="TinvEstados"}
  {cwbarrasuppanel titulo="Estados de los bienes"}
    {cwbotontooltip label="Insertar registros" accion="insertar"
actuaSobre="tabla"}
    {cwbotontooltip label="Modificar registros" accion="modificar"
actuaSobre="tabla"}
    {cwbotontooltip label="Eliminar registros" accion="eliminar"
actuaSobre="tabla"}
  {/cwbarrasuppanel}
  {cwcontenedor}
    {cwtabla conCheck="true" conCheckTodos="true" datos=$smtty_datosTabla}
      {cwfila tipoListado="false"}
        {cwcampotexto nombre="lisCodEstado" label="Cód. Estado."
editable="nuevo" size="2" value=$defaultData_TinvEstados.lisCodEstado dataType=
$dataType_TinvEstados.lisCodEstado}
        {cwcampotexto nombre="lisDescEstado" label="Desc. Estado."
editable="true" size="12" value=$defaultData_TinvEstados.lisDescEstado dataType=
$dataType_TinvEstados.lisDescEstado}
      {/cwfila}
      {cwpaginador enlacesVisibles="3"}
    {/cwtabla}
  {/cwcontenedor}
```

```
{cwbarrainfpanel}
  {cwboton id="BtnGuardarEdi" iconCSS="glyphicon glyphicon-ok" label="Guardar"
class="button" accion="guardar" }
  {cwboton id="BtnCancelarEdi" iconCSS="glyphicon glyphicon-remove" label="Cancelar"
class="button" accion="cancelar" }
{/cwbarrainfpanel}
{/cwpanel}
{/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 [124].

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 aplicación puede dividirse en módulos, etiqueta `<modulo>`, aunque normalmente sólo lo harán aquellas de tamaño grande; una aplicación 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:

Las 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 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 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 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 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:

- **título:** 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 menú

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, sería 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 sería 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 sería 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). Así 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 sería 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 desactivación 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 equivalente.

Smarty [https://www.smarty.net/about_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 una plantilla (tpl) básica.

Los elementos que aparecen entre llaves dentro de una plantilla (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. Ejemplo:

```
{cwtabla conCheck="true" seleccionUnica="true" datos=$smarty_datosTabla}
  {* ... *}
{/cwtabla}
```

Ej. {**cwtabla** conCheck="true" seleccionUnica="true" datos=\$smarty_datosTabla}

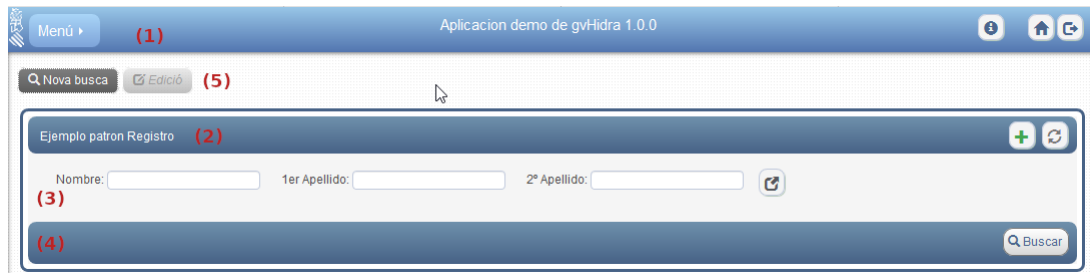
...

{/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" label="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 Anatomía 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 hijos 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} (3)**

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

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

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 (LIS) 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 (EDI) Necesitaremos insertar el plugin **{cwfechaedicion}** 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 (FIL) 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} (5)

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

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 [221] 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 principal {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 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}

{* 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}

{/cwmarcopanel}
{/cwventana}

```

3.4.3. Diseño avanzado de plantillas.

La versión 5 de gvHIDRA introduce cambios fundamentales en el uso del motor de plantillas Smarty, en gran parte debido a que se hace uso de la versión 3 de Smarty, la cual nos permite mayor flexibilidad, expresividad y rendimiento.



Nota

gvHIDRA hace uso ahora de Smarty 3 de forma nativa, y **no en modo de compatibilidad hacia atrás**. Por tanto, la clase `Smarty_Phrame`, utilizada por gvHIDRA para extender el comportamiento del motor de plantillas, **hereda de la clase `Smarty`** y no hereda de `SmartyBC` (Smarty Backward Compatibility).

Existen en el framework tres carpetas de plugins, dos de ellas corresponden a plantillas nativas de Smarty y una tercera corresponde a los plugins propios de gvHIDRA.

1. En `igep/smarty/sysplugins`, se encuentran los plugins del core de Smarty.
2. En `igep/smarty/plugins`, se encuentran los plugins estándar de Smarty que pueden ser utilizados (así como librerías externas que se han incorporado al framework).

3. En `igep/smarty/gvhplugins`, se encuentran los plugins desarrollados expresamente para el framework gvHIDRA.

Opcionalmente, pueden existir dos directorios más de plugins, uno de ellos correspondiente a plantillas del custom actual y otro directorio que corresponde con las plantillas personalizadas de la aplicación

4. En `custom/xxx/smarty/plugins`, se encuentra la personalización de los plugins del custom actual.

5. En `smarty/plugins`, se encuentran la personalización de los plugins la aplicación actual.

Éstos dos últimos directorios pueden no existir, especialmente el de la aplicación, ya que no son estrictamente necesarios y solo sirven para modificar el comportamiento por defecto de los plugins existentes, o para añadir nuevos plugins al custom y/o a la aplicación.

La precedencia en la definición de los plugins siempre es de más cercana a más lejana. Es decir, un mismo plugin definido en la carpeta de aplicación sobrescribirá el comportamiento que se haya definido para el mismo plugin en la carpeta custom, y a su vez sobrescribirá al comportamiento definido para la carpeta plugin del framework gvHIDRA, que a su vez sobrescribirá al plugin nativo de Smarty.

Por tanto, el orden de precedencia en las carpetas de plugins es:

```
smarty/plugins
> custom/xxx/smarty/plugins
> igep/smarty/gvhplugins
> igep/smarty/plugins
> igep/smarty/sysplugins
```

...

INDICACIONES DEL USO DE `{include...}` `{block...}` .

Ubicación de botonera "nueva búsqueda", "listado", "edición" .

3.4.4. 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 [221].



Nota

A diferencia de versiones anteriores de gvHIDRA, la versión 5.x obliga a escribir los **nombres de los plugins en minúsculas** cuando son utilizados en las plantillas (TPL).

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 [21].

3.5.1. Operaciones y métodos virtuales

Vamos a adentrarnos 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*) sobrescribiendo 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 [65]
2. preBuscar [71], postBuscar [71]
3. preInsertar [66], postInsertar [66]
4. preNuevo [68]
5. preModificar [68], postModificar [68]
6. preBorrar [69], postBorrar [69]
7. preEditar [69], postEditar [69]
8. preRecargar [70], postRecargar [70]
9. openApp [71], closeApp [71]

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`

En el caso de querer obtener el valor o asignar el valor de un elemento "external", se tendrá que invocar de la siguiente forma:

```
// Obtener el valor de un elemento 'external'
$objDatos->getValue ('nombreCampo', 'external');

// Asignar el valor de un elemento 'external'
$objDatos->setValue ('nombreCampo', 'nuevoValor', 'external');
```

- `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 [76]):

//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 sobrescribir 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&tipoAcceso=A"/>
```

```
<opcion titulo="Ventana Con Modificar" descripcion="Permitimos modificar"
url="phrame.php?action=Clase__iniciarVentana&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 clase `Manejadora`:

```
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('smtty_administrador', $admin);
```

Ahora en la tpl utilizaremos los métodos de la librería IgepJS.js (ver punto D.6.7):

```
{cwcampotexto nombre="cif" editable="true" size="5" label="CIF"}
{cwcampotexto nombre="nombre" editable="true" size="40" label="Nombre"}
{cwcampotexto nombre="esAdministrador" oculto="true"}
...
<script defer>
$(document).ready( function() {
gvh.subscribeRewireUI("ClaseManejadora",function() {
var objIgepJS = new gvh.IgepJS('ClaseManejadora','edi');
if( objIgepJS )
{
var esAdministrador = objIgepJS.getValue('esAdministrador');
if (esAdministrador == 'Administrador')
{
objIgepJS.setVisible('cif',true);
objIgepJS.setVisible('nombre',true);
}
else {
objIgepJS.setVisible('cif',false);
objIgepJS.setVisible('nombre',false);
}
}
} );
} );
</script>
```

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[$scentroSeleccionado]);
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 sobrescribir 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()** [43].

La acción de buscar, tal y como se ha comentado en el punto anterior Operaciones y métodos virtuales [71], 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 caracteres 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 búsqueda 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 caracteres 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 seleccionar 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 label="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 pérdida 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" label="Cif"}
&nbsp;
{cwcampotexto nombre="orden" editable="true" size="2"
  label="Orden" actualizaA="nombre"}
```

```
&nbsp;{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 cwstring [241].
- **setSelected(\$campo,\$valor)**: Cambia el valor seleccionado de una lista.
- **setFocus(\$campo)**: Fija el foco en el elemento.
- **setEnabled(\$campo,\$valor)**: Cambia la accesibilidad de un campo, también se puede aplicar este método a un elemento cwstring [241] cuando es de tipo enlace.
- **setIcon(\$campo,\$valor)**: Cambiará el icono css del campo. Método que solamente actúa sobre componentes del tipo cwstring [241], 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 cwstring [241] 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->setEnabled('nfactura',true);
$objDatos->setEnabled('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

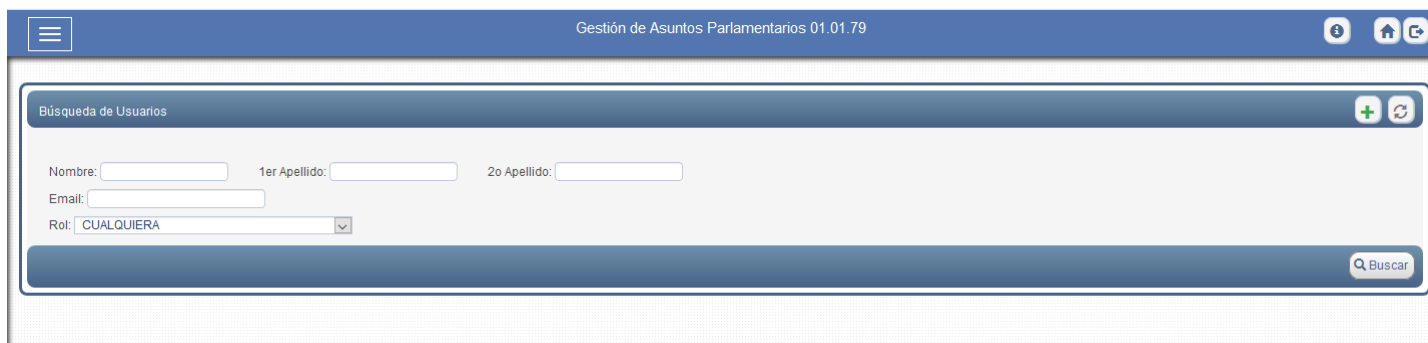
El framework suministra un método para personalizar su aspecto y funcionamiento, que en adelante llamaremos 'temas'. Los temas están ubicados en la carpeta en **app/custom**, podemos tener varios temas y activar uno en concreto mediante el atributo **customDirName** del fichero **gvHidraConfig.inc.xml** del framework o de la aplicación, o con el método **ConfigFramework->setCustomDirName** (ver configuración de **gvHidraConfig.xml** [28]).

Con gvHidra vienen estos temas configurados:

- Custom **greyStyle**

Es el estilo clásico de gvHidra, el que se venía distribuyendo hasta ahora.

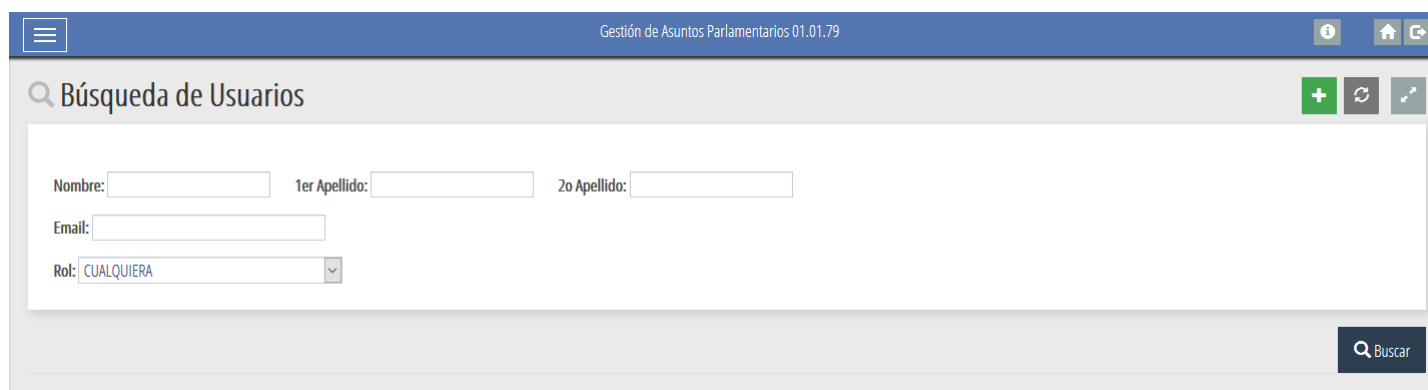
Figura 3.1. Custom greyStyle



- Custom **lightStyle**

Se ha creado una versión del custom con colores más claros, elementos más grandes y espaciados entre sí. Se incluye una versión compact de este estilo: **cpLightStyle**

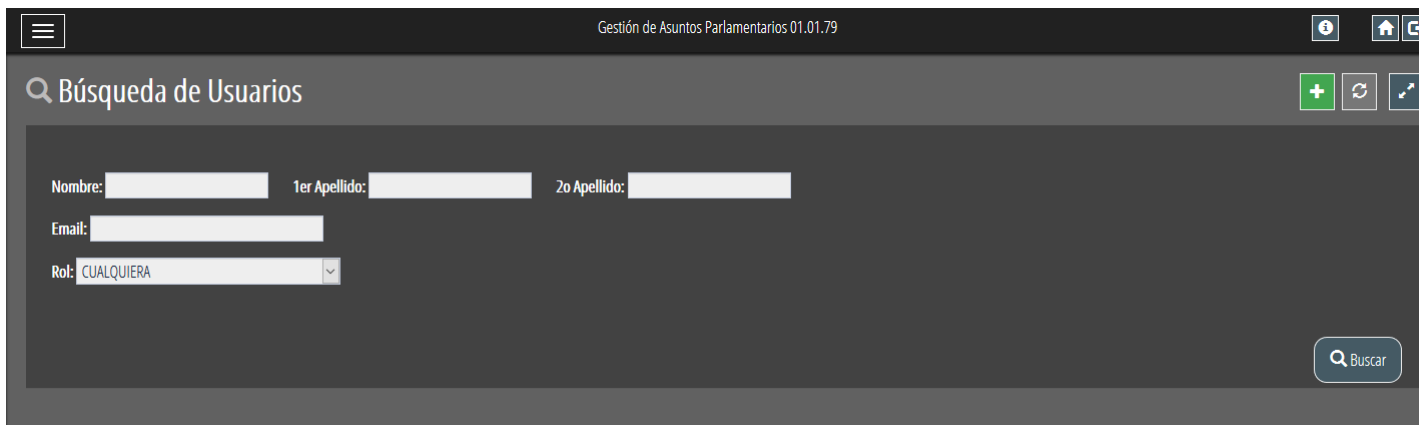
Figura 3.2. Custom lightStyle



- Custom **darkStyle**

El darkStyle es una versión, como su propio nombre indica, más oscura del custom con tonalidades grises y negro, elementos más grandes y espaciados entre sí. Se incluye una versión compact de este estilo: **cpDarkStyle**

Figura 3.3. Custom darkStyle



El tema de los customs está más desarrollado en el apartado **7.4 Customs [174]**.

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 usuario 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"
  label="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**
6. **gvHidraTime**

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.

- '#' Indica una letra (mayúscula o minúsculas)

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

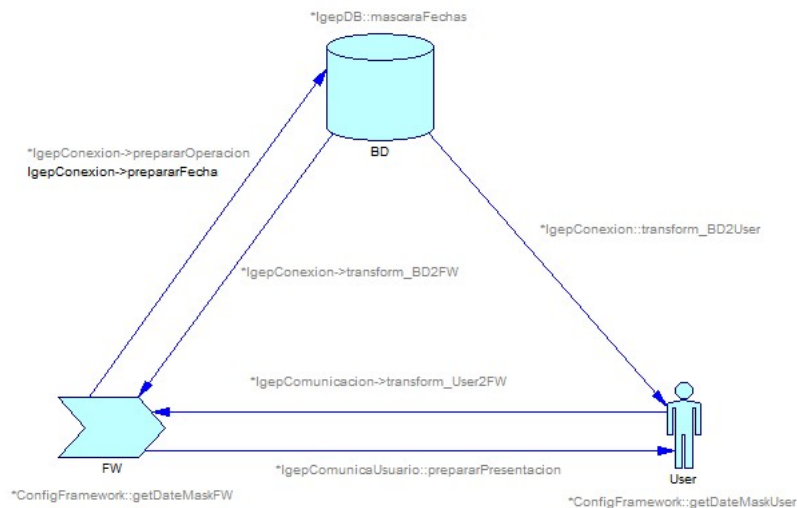
La intención de gvHidra es simplificar lo máximo posible los problemas derivados de los formatos de las fechas, y para ello se ofrece al programador una forma única de manejar las fechas (formato de negocio) y es el framework el que se encarga, en caso necesario, de convertir estas fechas al formato 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 IgepDB-MS_*) 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 gvHydraDate o gvHydraDatetime 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 gvHydraDate(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 DateTime (DateTime [<http://php.net/manual/es/book.datetime.php>]) (si no tiene valor se recibirá un NULL).

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

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

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

La clase DateTime 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 días 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 día 31 y le sumamos un mes, si no existe el día 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 debería usarse para obtener elementos de la fecha como el día 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 es 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 intervalo [f1, f2].
- **betweenDays(\$f1, \$days)**: comprueba si la fecha del objeto se encuentra entre el intervalo [f1, (f1 + days)]. Si el número de días 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, así 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 día 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 produciría 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 DateTime 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 24/02/2019 13:28:30

$f = new DateTime('now');
print($f->format("d/m/Y H:i:s")); // equivalente a $f->formatUser()
```

```
//Resultado = 24/02/2019 13:28:30

$f = new DateTime("today");
print($f->format("d/m/Y H:i:s"));
//Resultado = 24/02/2019 00:00:00

$f = new DateTime("yesterday");
print($f->format("d/m/Y H:i:s"));
//Resultado = 23/02/2019 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 DateTime("today") < $fpeticionIni)
```

Asignar una fecha cualquiera:

```
$f = new DateTime();
$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 días 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

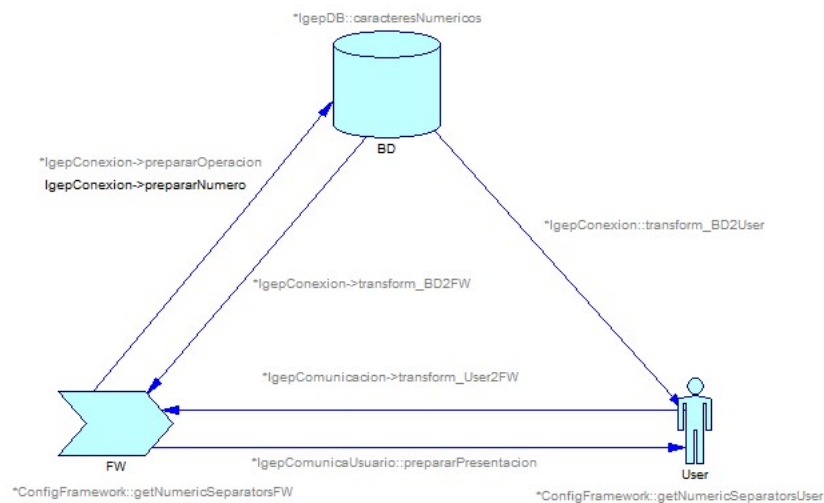
Cuando trabajamos con números podemos encontrarlos con los mismos problemas que hemos visto para las fechas, y que son los que vienen derivados de los distintos formatos que pueden usarse para representarlos.

En gvHIDRA se definen los formatos numéricos indicando el caracter usado como separador decimal, y el separador de miles (puede ser cadena vacía). 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 **IgepDB-MS***. 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 carácter (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 método `setValue()` del objeto datos usando un número 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

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" label="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 PostgreSQL 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 usarán 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($conn=null)
    {

    }
}
```

```
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 constructor de la clase recibe un objeto conexión correspondiente al objeto de conexión de la clase manejadora. Esta nueva propiedad (activa desde la versión 5.0.0), mejora el rendimiento ya que evita que en cada instancia de clase se realice una conexión propio.
- El método **build**, devuelve como resultado un *array*. Este array, puede ser array vacío 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 usarán 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 cláusulas `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" label="Provincia" dataType=
  $dataType_claseManejadora.codigoProvincia datos=
  $defaultData_claseManejadora.codigoProvincia}

{cwlista nombre="codigoMunicipio" size="3" editable="true" label="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:

- **getList**: obtenemos la lista con la que queremos trabajar.
- **clean**: limpiamos los valores dejándola vacía.
- **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 **gvHidraCheckBox** y de sus métodos, 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" label="Coche"
  dataType=$dataType_claseManejadora.filCoche
  valor=$defaultData_claseManejadora.filCoche}
```

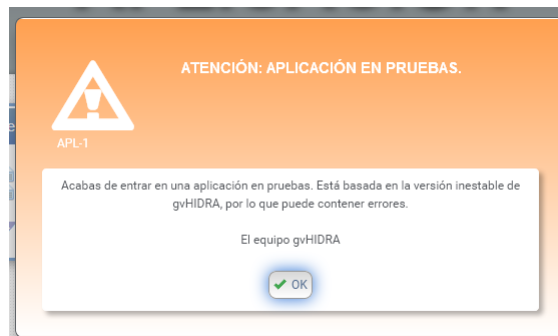
Es importante añadir el parámetro `dataType` que es el que asociará la definición dada en la clase manejadora con el plugin.

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.



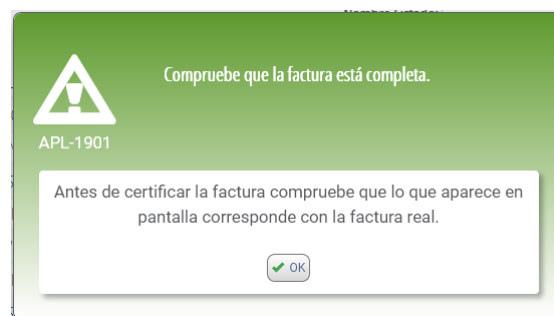
3. Errores.

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 relleno 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 `cwboton` (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.

Ejemplo de botón guardar, al pulsar "*Si*" se ejecutará la acción guardar del panel

```
{cwboton label="Guardar" class="boton" accion="guardar" confirm="APL-5" }
```

Ejemplo de botón con acción particular, al pulsar "*Si*" se ejecutará la acción particular `accionConfirmar`. Esta acción debe estar definida como una acción particular (punto 3.5.3.2) en la clase manejadora.

```
{cwboton label="Confirmar acción" class="boton" accion="particular"
  action="accionConfirmar" confirm="APL-5" }
```



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 útil 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 `value` 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_ano` para la clase `Manejadora Presupuesto`.

```
//en la clase manejadora
$this->addDefaultData ('fil_ano', '2012');

...

{*en la tpl*}
{... value=$defaultData_Presupuesto.fil_ano...}
```

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, 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. Las pestañas de los detalles pueden ahora mostrar el n.º de tuplas que devolvería la consulta al pinchar en cada pestaña.

```
public function addSlave( $nombreClaseManejadora, $listasCamposPadre,
    $listasCamposHijo, $contarTotalTuplas=false, $agruparConteo=true )
```

Los parámetros que requiere este método son los siguientes:

- **\$nombreClaseManejadora:** Nombre de la clase manejadora del detalle.
- **\$listasCamposPadre:** Array de campos correspondientes al maestro que se relacionan con el detalle.
- **\$listasCamposHijo:** Array de campos correspondientes al detalle que se relacionan con el maestro.
- **\$contarTotalTuplas:** Campo booleano opcional con el que se indicará si se quiere el conteo o no en las pestañas del detalle. Por defecto es false.

Para el conteo se construirá una consulta que por defecto utilizará las `searchQueries()` definidas en las distintas clases detalle/hija (definidas en el constructor de cada clase manejadora con los métodos `'setSelectForSearchQuery'` y `'setWhereForSearchQuery'`). Si quisiera optimizarse la consulta de conteo en cualquiera de los detalles, se puede utilizar un nuevo método `'setSelectForDetailCount'`, de funcionamiento similar a `'setSelectForSearchQuery'` pero que se usará solo para el conteo de tuplas al actuar como detalle/hija. Si no se especifica nada en `'setSelectForDetailCount'`, se usará lo especificado en `'setSelectForSearchQuery'`. En cuanto a las condiciones de filtro de la búsqueda (aka. sección sql WHERE), se utilizarán las mismas que para la consulta de búsqueda normal, es decir, se usarán las especificadas en `'setWhereForSearchQuery'` y en los filtros. De ese modo, garantizaremos que se filtran siempre las mismas tuplas en el conteo que en la búsqueda.

- **\$agruparConteo:** Campo booleano opcional que agrupa la consulta de conteo de tuplas junto con la consulta de otros detalles. Por defecto es true.

```
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*. Debemos 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', "smtty_datosTablaM");
$panelMaestro->activarModo("fil", "estado_fil");
$panelMaestro->activarModo("lis", "estado_lis");
$comportamientoVentana->agregarPanel($panelMaestro);

// Definición del DETALLE
$panelDetalle = new IgepPanel('TinvSubtipos', "smtty_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"**)


```

{cwpanel id="lis" esMaestro="true" itemSeleccionado=$smt_y_filaSeleccionada
action="operarBD" method="post" estado=$estado_lis claseManejadora="TinvTipos"}
  {cwbarrasuppanel titulo="Tipos de Bienes"}
    {cwbotontooltip title="Insertar registros" accion="insertar"}
actuaSobre="tabla" ocultarMD="Detalle"}
  {cwbotontooltip title="Modificar registros" accion="modificar"}
actuaSobre="tabla"}
  {cwbotontooltip title="Eliminar registros" accion="eliminar"}
actuaSobre="tabla"}
  {cwbotontooltip title="Limpiar Campos" accion="limpiar" actuaSobre="tabla"}
  {/cwbarrasuppanel}
  {cwcontenedor}
  {cwtabla conCheck="true" conCheckTodos="false" numFilasPantalla="4" datos=
$smt_y_datosTablaM}
    {cwfila tipoListado="false"}
      {cwcampotexto nombre="lisCodigoTipo" editable="nuevo" size="2"
label="Cod.Tipo" dataType=$dataType_TinvTipos.lisCodigoTipo}
      {cwcampotexto nombre="lisDescTipo" editable="true" size="25" label="Tipo"
dataType=$dataType_TinvTipos.lisDescTipo}
    {/cwfila}
    {cwpaginador enlacesVisibles="3"}
  {/cwtabla}
  {/cwcontenedor}
  {cwbarrainfpanel}
  {cwboton label="Guardar" class="boton" accion="guardar"}
  {cwboton label="Cancelar" class="boton" accion="cancelar"}
  {/cwbarrainfpanel}
{/cwpanel}

</td></tr>
<tr><td>
<!-- ***** PANEL DETALLE *****-->
  {if count($smt_y_datosTablaM) gt 0}
    {cwpanel tipoComprobacion="envio" action="operarBD" detalleDe="lis" method="post"
estado="on" claseManejadora="TinvSubtipos"}
      {cwbarrasuppanel titulo="Subtipos de Bienes"}
        {cwbotontooltip title="Insertar registros" accion="insertar"}
actuaSobre="ficha"}
        {cwbotontooltip title="Modificar registros" accion="modificar"}
actuaSobre="ficha"}
        {cwbotontooltip title="Eliminar registros" accion="eliminar"}
actuaSobre="ficha"}
      {/cwbarrasuppanel}
      {cwcontenedor}
      {cwfichaedicion id="FichaDetalle" datos=$smt_y_datosFichaD}
        {cwficha}
          <br /><br />
          {cwcampotexto nombre="ediCodigoSubtipo" editable="nuevo" size="3"
label="Código" dataType=$dataType_TinvSubtipos.ediCodigoSubtipo}
          <br /><br />
          {cwcampotexto nombre="ediDescSubtipo" editable="true" size="35"
label="Descripción" dataType=$dataType_TinvSubtipos.ediDescSubtipo}
          <br /><br />
          {cwcampotexto nombre="ediCodigoTipo" oculto="true" value=
$defaultData_TinvSubtipos.ediCodigoTipo}
        {/cwficha}
      {cwpaginador enlacesVisibles="3"}
    {/cwpanel}
  {/if}

```

```

    {/cwtabla}
  {/cwcontenedor}
  {cwbarrainfpanel}
    {cwboton label="Guardar" class="boton" accion="guardar"}
    {cwboton label="Cancelar" class="boton" accion="cancelar"}
  {/cwbarrainfpanel}
  {/cwpanel}
{/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_tipossubtipo.php&panel=listar');
$this->_AddForward('TinvTipos__recargar', 'gvHidraError', 'index.php?view=views/
patronesMD/M(FIL-LIS)-D(LIS)/p_tipossubtipo.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)

4.1.1.1.2. 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 queremos 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 *smt_y_panelActivo* el nombre de la clase manejadora que queremos.

```
<?php  
//MAESTRO  
$comportamientoVentana= new IgepPantalla();  
$panelMaestro = new IgepPanel('expedientes','smt_y_datosTablaM');  
$panelMaestro->activarModo("fil","estado_fil");  
$panelMaestro->activarModo("edi","estado_edi");  
$datosPanel = $comportamientoVentana->agregarPanel($panelMaestro);  
  
//DETALLE  
$panelDetalle = new IgepPanel('informes','smt_y_datosInformes');  
$panelDetalle->activarModo("lis","estado_lis");  
$datosPanelDetalle = $comportamientoVentana->  
>agregarPanelDependiente($panelDetalle,"expedientes");  
  
$panelDetalle = new IgepPanel('deficiencias','smt_y_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"  
    )  
);  
$s->assign('smt_y_detalles',$detalles);  
  
$s->assign('smt_y_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=$smtyp_tipoAviso codAviso=$smtyp_codError descBreve=
$smtyp_descBreve textoAviso=$smtyp_textoAviso onLoad=$smtyp_jsOnLoad}
  {cwbarra usuario=$smtyp_usuario codigo=$smtyp_codigo customTitle=$smtyp_customTitle}
  {cwmenulayer name="$smtyp_nombre" cadenaMenu="$smtyp_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=
$smtyp_filaSeleccionada action="operarBD" method="post" estado="$estado_edi"
claseManejadora="expedientes" accion=$smtyp_operacionFichaexpedientes}
  ...
  {/cwpanel}

  <!-- ***** PANELES DETALLES *****-->
20 </td></tr>
  {if count($smtyp_datosTablaM ) gt 0 }
  {cwdetalles claseManejadoraPadre="expedientes" detalles=$smtyp_detalles
panelActivo=$smtyp_panelActivo}
  <tr><td>

25 <!-- ***** Detalle 1: INFORMES *****-->
  {if $smtyp_panelActivo eq "informes"}
  {cwpanel id="lisDetalle" tipoComprobacion="envio" action="operarBD"
method="post" detalleDe="edi" estado="on" claseManejadora="informes"}
  ...
  {/cwpanel}
30 {/if}

  <!-- ***** Detalle 2: DEFICIENCIAS *****-->
  {if $smtyp_panelActivo eq "deficiencias" }
  {cwpanel id="ediDetalle" tipoComprobacion="envio" action="operarBD"
method="post" detalleDe="edi" estado="on" claseManejadora="deficiencias"}
35 ...
  {/cwpanel}
  {/if}

  {/if}
40 {/cwmarcopanel}
  {/cwventana}

```

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 origen. 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 simultáneamente.

```
$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');
...
```

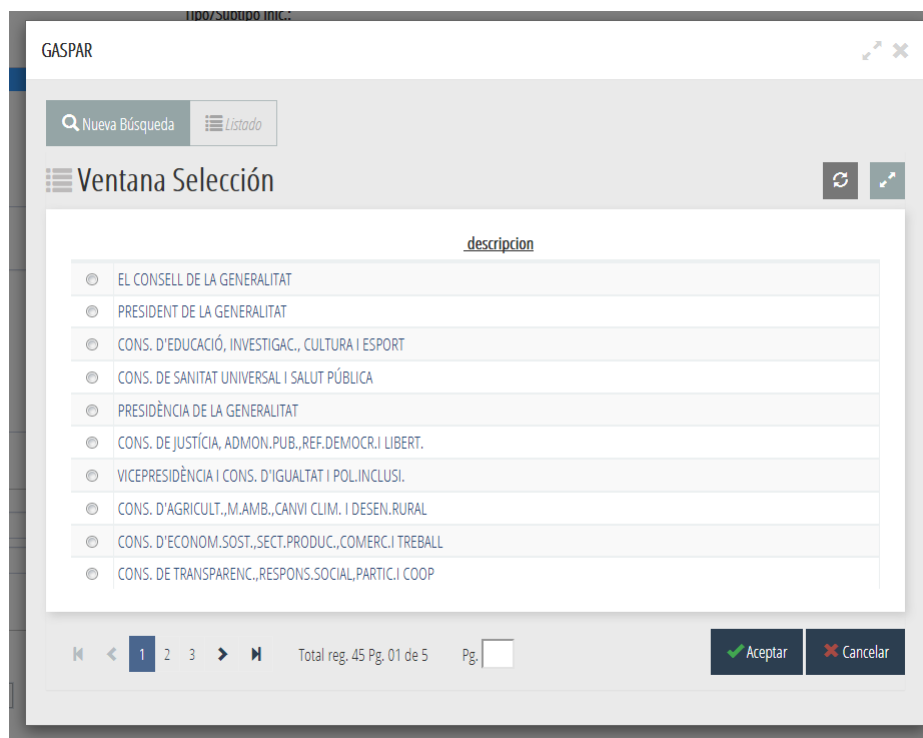
Si utilizamos este método significa que tenemos una plantilla (tpl) que dará formato a la ventana de selección. En caso contrario, el framework dibuja una ventana de selección sencilla, en la que su contenido no es configurable.

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.

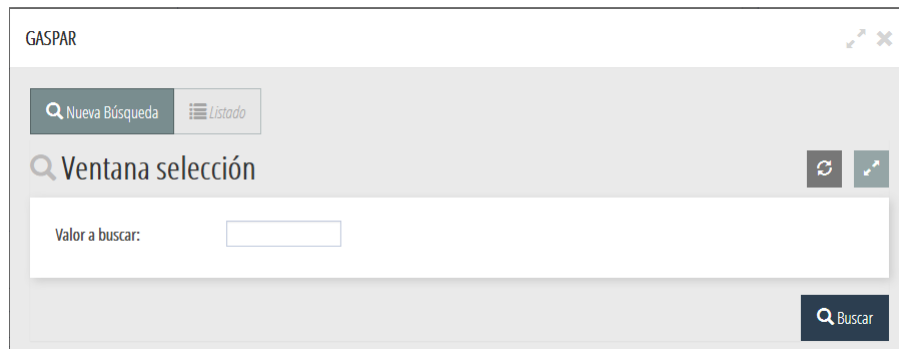
```
{cwcampotexto nombre="nombre" size="15" label="Nombre"}
{cwcampotexto nombre="descripcion" size="40" label="Descripción"}
{cwcampotexto nombre="id" oculto="true"}
```

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 actúa (en nuestro ejemplo usuario), el form y el panel. Aquí tenemos un ejemplo:

```
{cwcampotexto nombre="filUsuarios" size="8" label="Usuario"}
{cwbotontooltip 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. Visor de Mapas

Este componente surge por la necesidad de visualizar un mapa para ubicar un edificio, zona, yacimiento... en determinadas aplicaciones. Por lo tanto con el plugin `cwmaps` permitirá el acceso un visor de mapas centrado en unas coordenadas que se le indiquen. El plugin `cwmaps` dibuja un botón en el panel que esté incluido. Ese botón necesita de unos parámetros (coordenadas y acción) para realizar la llamada al visor. Al pulsar este botón se abrirá una capa donde se mostrará el visor del mapa con el punto seleccionado y/o polígono, según se decida en la programación.

Para el uso del visor se requiere incluir el plugin `cwmaps` en la tpl, que hará referencia a una acción particular definida en la clase manejadora que ejecutará el visor con las condiciones que se le indiquen utilizando los métodos de la clase `gvHidraMapaConfigGenerator`.

4.2.2.1. Clase gvHidraMapaConfigGenerator

Clase que permite generar la configuración JSON para visualizar el mapa. Esta clase contiene los siguientes métodos:

- `setBaseLayer($nombreCapa, $URL, $layers, $transparente = true, $format = "image/png")`

Fija el contenido de la capa base.

- **nombreCapa:** Nombre que le queramos dar a la capa y aparecerá en el visor.
- **url:** URL al servicio web de mapas correspondiente.
- **layers:** Capa que será la base del visor.
- `addLayer($nombreCapa, $URL, $layers, $opacity, $transparente = true, $format = "image/png")`

En el caso de que queramos añadir capas extra a la capa base, definiremos sus propiedades.

- **nombreCapa:** Nombre que le queramos dar a la capa y aparecerá en el visor.
- **url:** URL al servicio web de mapas correspondiente.
- **layers:** Capa que será la base del visor.
- **oacity:** Opacidad de la capa sobre la capa base.
- `setGeometry($contenidoWKT, $proyeccion = 'EPSG:4326', $centrar = true)`

En el caso de una representación tipo polígono, con este método se establecerá el contenido de la geometría.

- **contenidoWKT:** Coordenadas que delimitan el polígono (p.ej: POLYGON(((695269 4247375,695528 4247595,695416 4247726,695187 4247550)))

- **proyeccion:** Código EPSG
- **setPoint(\$latitud, \$longitud, \$proyeccion = 'EPSG:4326', \$centrar = true)**

En el caso de una representación tipo putno, con este método se establecerá el contenido del punto.

- **latitud:** Coordenada Y
- **longitud:** Coordenada X
- **proyeccion:** Código EPSG
- **toJSON()**

Método que devolverá un JSON con toda la información dada a través de los métodos anteriores.

4.2.2.2. Plugin cwmaps (tpl)

Para el uso del visor de mapas, lo primero que hay que hacer es incluir el plugin cwmaps en la plantilla (tpl). También es necesario tener en la ventana los campos correspondientes a las coordenadas X e Y. El plugin tiene los siguientes parámetros específicos que nos permitirán definir el contexto del visor:

- **coordX:** Corresponderá al nombre del campo de la plantilla que contendrá la coordenada X.
- **coordY:** Corresponderá al nombre del campo de la plantilla que contendrá la coordenada Y.
- **zoom:** Indica el zoom que se aplicará a la entrada del mapa. Valores más pequeños menor zoom, valores más grandes mayor zoom (por defecto 10).
- **action:** Corresponderá a la acción particular que se tiene que definir en la clase manejadora correspondiente.

```
{cwcampotexto label="Coordenada X" nombre="edi_AR_UTMN" size="20" editable="true"
visible="true" value=$defaultData_ClaseManejadora.edi_UTMN dataType=
$dataType_ClaseManejadora.edi_IN_UTMN}
{cwcampotexto label="Coordenada Y" nombre="edi_AR_UTME" size="20" editable="true"
visible="true" value=$defaultData_ClaseManejadora.edi_IN_UTME dataType=
$dataType_ClaseManejadora.edi_IN_UTME}
{cwmaps nombre="visor" coordX="edi_AR_UTME" coordY="edi_AR_UTMN"
zoom="12" action="Arqueologicos__poligono" label="MAPA" }
```



4.2.2.3. Acción particular en la clase manejadora

En la acción particular de la clase manejadora se definirán todos los valores para poder obtener el polígono y/o punto del mapa a visualizar. Para ello se invoca a la clase gvHidraMapConfigGenerator para ir definiendo los parámetros, tal y como se indica en el siguiente ejemplo:

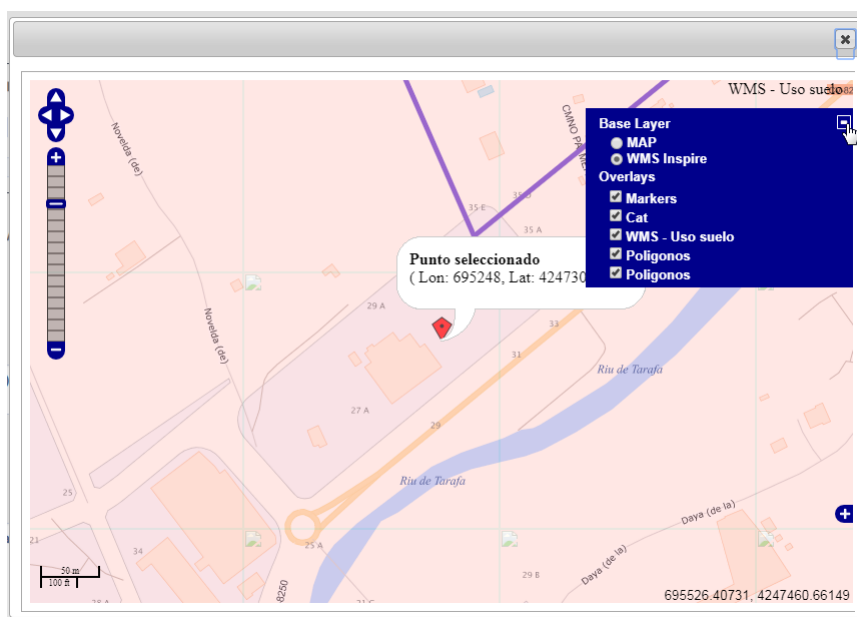
```
public function accionesParticulares($str_accion, $objDatos) {
switch($str_accion) {
case 'poligono':
    $tupla = IgepSession::dameTuplaSeleccionada('ClaseManejadora');
    // Obtenemos los datos que permitirán definir el polígono y punto en el
```

mapa

```

    $sql = "SELECT AR_DELIMITACION AS "DELIMITACION", AR_UTMN AS "UTMN",
AR_UTME AS "UTME" FROM ARQUEOLOGICOS WHERE ID={$stupla['ID']}";
    $res = $this->consultar($sql);
    $wktPoligono = $res[0]['DELIMITACION'];
    $UTMN = $res[0]['UTMN'];
    $UTME = $res[0]['UTME'];
    // Invocamos la clase gvHidraMapConfigGenerator para configurar los datos para
el mapa. Esta clase nos devolverá un JSON con los datos de configuración.
    $gen = new gvHidraMapConfigGenerator();
    // Establecemos el contenido de la capa que será la base
    $gen->setBaseLayer("WMS Inspire", "http://www.ign.es/wms-inspire/ign-base",
"IGNBaseTodo");
    // Establecemos el polígono a mostrar
    if(!empty($stupla)) {
        $gen->setGeometry($wktPoligono, 'EPSG:25830');
    }
    // Establecemos el punto a mostrar al que se le pasará la longitud, latitud y
proyección, para poder visualizar tanto punto como polígono en el mapa, los dos a la
vez.
    $gen->setPoint($UTMN, $UTME, "EPSG:25830");
    // Opcional: si se quieren añadir capas con información extra.
    $gen->addLayer("Cat", "http://ovc.catastro.meh.es/Cartografia/WMS/
ServidorWMS.aspx", "Catastro", 0.35);
    $gen->addLayer("WMS - Uso suelo", "http://www.ign.es/wms-inspire/ign-base",
"LU.ExistingLandUse", 0.35);
    // Obtenemos el JSON con la información a mostrar en el mapa
    $jsonToSend = $gen->toJSON();
    ob_clean();
    print $jsonToSend;
    die;
break;
default:
    break;
}
}

```



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 programador 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 PostgreSQL. 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 raíz de nuestro proyecto (también se podría 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" label="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 necesitaríamos 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 petición 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 continúe 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 perderíamos 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" label="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".

```

{cwbarrasuppanel}
  {cwbotontooltip titulo="+" accion="saltar" actuaSobre="ficha" }
{/cwbarrasuppanel}

```

- 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" label="CIF" dataType=
$smty_dataType_Personas.ediCif}
  {cwbotontooltip 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 establece 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" label="ACUMULAR" class="boton" accion="volver"}
{cwboton id="cancelarAcumular" label="CANCELAR" class="boton" accion="volver"}
```

Este estímulo se recibirá en la clase objeto del salto (en nuestro ejemplo Aspirantes). En ella debemos sobrescribir 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 tendríamos 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, queremos 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" label="CIF" dataType=
$smty_dataType_Facturas.ediCif}
{cwbotontooltip id="addProveedor" 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 `smtj_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 $smtj_modal neq 1}
  {cwpanel id="fil" action="buscar" method="post" estado="$estado_fil"
  claseManejadora="Proveedores"}
  {cwbarrasuppanel titulo="Mantenimiento de proveedores"}
  ...
{/if}
<!-- ***** PANEL edi *****-->
  {cwpanel id="edi" tipoComprobacion="envio"
  action="$smtj_operacionFichaProveedores"
  method="post" estado="$estado_edi" claseManejadora="Proveedores" accion=
  $smtj_operacionFichaProveedores}
  ...
  {cwbarrainfpanel}
  {cwboton label="Guardar" class="button" accion="guardar"}
  {*Si la abrimos como modal que al cancelar cierre la ventana. Sino que se
  comporte normal*}
  {if $smtj_modal eq 1}
    {cwboton label="Cancelar" class="button" accion="cancelar"
  action="cancelarModal"}
  {else}
    {cwboton label="Cancelar" class="button" accion="cancelar"
  action="cancelarEdicion"}

```



```

    {/if}
{/cwbarrainfpanel}
...

```

Nota: Este cambio es opcional, es para ajustarse a las necesidades del ejemplo.

Arriba vemos que se definen dos botones (cwboton), uno para cuando la ventana es modal y otro para cuando no lo es. En el caso de encontramos 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 código 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 identificación 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 resuelve su ejecución.

```
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	127
5.1. Conexiones BBDD	127
5.1.1. Bases de Datos accesibles por gvHidra	127
5.1.2. Acceso y conexión con la Base de Datos	127
5.1.3. Transacciones	133
5.1.4. Procedimientos almacenados	134
5.1.5. Recomendaciones en el uso de SQL	135
5.2. gvHidraDBAL	136
5.2.1. API de gvHidraDBAL	136
5.3. Web Services	148
5.3.1. Web Services en PHP	149
5.3.2. Web Services en gvHIDRA. WSComun	150
6. Seguridad	151
6.1. Autenticación de usuarios	151
6.1.1. Introducción	151
6.1.2. Crear un nuevo método de Autenticación	152
6.2. Modulos y Roles	155
6.2.1. Introducción	155
6.2.2. Uso en el framework	157

Capítulo 5. Fuentes de datos

5.1. Conexiones BBDD

5.1.1. Bases de Datos accesibles por gvHidra

Históricamente, el lenguaje PHP ha contado con multitud de DRIVERS, a través de los cuales se ha permitido la conexión e interogación de diversos sistemas de gestión de Base de Datos (SGBDs), tanto los relacionales (ORACLE, PostgreSQL, MySQL / MariaDB, SQLite, MSSQL Server...) como los noSQL o no relacionales (MongoDB, Cassandra...).

Cada uno de estos drivers "nativos" era más o menos similar a nivel funcional que el resto, en función de las similitudes de los SGBDs, sin embargo, a pesar de gran parte de la funcionalidad era comun (estándar SQL) cada uno de ellos mantenía su propia notación y funciones diferentes, es decir, su propia API, lo cual poco portable las aplicaciones si se decidía pues para cambiar de SGBD había que revisar todo el código fuente.

Para intentar solucionar estos aspectos, surgen distintos proyectos de abstracción de acceso a los datos, o en inglés DAL (Data abstraction Layer). Entre los mismos, destacan dos, el proyecto PEAR::MDB2 (actualmente abandonado) y el proyecto PDO (PHP Data Object) que es nativo del lenguaje PHP a partir de su versión 5.

En gvHidra, la gestión de las conexiones a base de datos se han delegado en la clase **IgepConexion**. Esta clase, propia de GVHidra, utiliza internamente tanto PEAR::MDB2 como PDO, y actúa como capa de abstracción, de forma que podemos acceder e interogar a diversas BBDDs sin preocuparnos por el tipo de la misma, ya que tenemos una API única.

Sin embargo, a pesar de usar la capa de abstracción (que unifica la API), sigue habiendo aspectos que dependen del SGBD: uso de características especiales (secuencias, limit, bloqueos, ...), manejo de transacciones, representación de fechas, de números (separadores de miles, decimales...) símbolos de moneda, el NULL como valor o estado, etc. Para resolver estas divergencias "finas", el framework define en su capa de abstracción diferentes funciones y métodos, que no siempre recaen sobre IgepConexion:

- 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 para limitar el número de registros de una consulta, las funciones de concatenación de cadenas, el uso de secuencias...
- Unificar el tratamiento de errores: por ejemplo, cuando hay un registro bloqueado, poderlo detectar independientemente de la BD.

Actualmente los SGBD probados con GVHidra son ORACLE (versiones 9 en adelante), PostgreSQL, MySql/MariaDB, SQLite, MS-SQL Server e Informix. Además la teoría dice que a través de PDO podrían soportarse muchos otros (Firebird, ODBC, DB2...).

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 patrón gvHidraForm_DB o el propio debug del Framework, (IgepDebug), son ejemplos de uso de conexiones a SGBD con API transparente para el programador. En ambos casos, gvHidra se encarga de conectar con el SGBD correspondiente y operar con él.

Evidentemente, seguro que nos encontramos 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 y PDO conecta con diferentes SGBD. Esto permite crear conexiones a diferentes fuentes de forma homogénea e independizarnos 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
Informix	IgepDBMS_informix.php
MicroSoft SQL Server	IgepDBMS_mssql.php y/o IgepDBMS_sqlsrv.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 se basa en la propuesta de MDB2, de forma que queda la siguiente estructura (array asociativo en PHP):

```
//Ejemplo DNS MDB2 para PostgreSQL
$dsnPos = array(
    'phptype' => 'pgsql',
    'username' => 'xxxx',
    'password' => 'xxxx',
    'hostspec' => 'xxxx',
    'database' => 'xxxx',
);

//Ejemplo DNS MDB2 para 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. Sin embargo, pese a ser compatibles con los ejemplos anteriores, cuando se carga la configuración desde fichero, se realizan adaptaciones de forma interna para facilitar la conexión.

Veamos por ejemplo, la correspondencia entre una entrada en el fichero de configuración, y la carga del array resultante cuya notación queda extendida con las aportaciones que realiza GVHidra

```
<dbDSN id='g_dsn' sgbd='oracle-thin' driver='pdo'>
  <dbHost>dbdsgbd.srv.gvhidra.es</dbHost>
  <dbPort>1521</dbPort>
  <dbDatabase>mybbdd</dbDatabase>
  <dbUser>EXUSER</dbUser>
  <dbPassword>xxxx</dbPassword>
</dbDSN>
```

```
array (
  'xmltype' => 'oracle-thin',
  'xmlhost' => 'dbdsgbd.srv.gvhidra.es',
  'xmlport' => 1521,
  'xmlldb' => 'predesa',
  'xmlcharset' => 'latin1',
  'phptype' => 'oci8',
  'username' => 'EXUSER',
  'password' => 'xxxx',
  'hostspect' => 'dbdsgbd.srv.gvhidra.es',
  'port' => 1521,
  'service' => 'DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=dbdsgbd.srv.gvhidra.es)
(PORT=1521))(CONNECT_DATA=(SID=mybbdd))',
  'driver' => 'PDO'
```

Veamos otro ejemplo, esta vez con conexión a Oracle RAC (Real Application Cluster de Oracle)

```
<dbDSN id='g_dsn' sgbd='oracle-rac' driver='MDB2'>
  <dbHost>dbdsgbd.srv.gvhidra.es</dbHost>
  <dbPort>1521</dbPort>
  <dbDatabase>mybbdd</dbDatabase>
  <dbUser>EXUSER</dbUser>
  <dbPassword>xxxx</dbPassword>
</dbDSN>
```

```
array (
  'xmltype' => 'oracle-rac',
  'xmlhost' => 'dbdsgbd.srv.gvhidra.es',
  'xmlport' => 1521,
  'xmlldb' => 'predesa',
  'xmlcharset' => 'latin1',
  'phptype' => 'oci8',
  'username' => 'EXUSER',
  'password' => 'xxxx',
```



```
'hostspec' => 'dbdsgbd.srv.gvhidra.es',
'port' => 1521,
'service' => '(DESCRIPTION=(LOAD_BALANCE=on)(ADDRESS=(PROTOCOL=TCP)
(HOST=dbdsgbd.srv.gvhidra.es)(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=predesa))',
'driver' => 'MDB2'
```

```
//Nueva conexion a postgresql
$conPos = new IgepConexion($dsnPos);
```

```
//Nueva conexion a oracle
$conOra = new IgepConexion($dsnOra);
```

¿Conexiones persistentes?

Si, algunos de los drivers con los SGBDs permiten crear conexiones persistentes, de forma que no es necesario "reconectar" con el SGBD cada vez, disminuyendo el "lag" de conexión, es decir, se mejora el rendimiento. Sin embargo, utilizar este tipo de conexión es peligroso en entornos y aplicaciones multiusuario (la mayoría), dado que el se comparte el contexto de la conexión (transacciones abiertas, formatos de fecha y hora, etc.). La recomendación es de forma general *NO UTILIZAR CONEXIONES PERMANENTES*, sin embargo, si en algún caso puntual, se quiere utilizar este tipo de conexiones podemos indicarlo a través de un parámetro pasado al constructor de la conexión (bool). Este parámetro, indica si el programador quiere solicitar una conexión persistente, siendo su valor por defecto *false*.

```
//Conexion permanente
$con = new IgepConexion($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 vemos 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 caracteres 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 sería 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 estén 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 nativo

En algunos casos, puede ser interesante trabajar directamente con el driver `PEAR::MDB2` nativo o el driver `PDO` de PHP. Por ejemplo, puede ser necesario en la invocación de procedimientos almacenados. Para estos casos la clase `IgepConexion()` FW ofrece el método `getConnection()` que devuelve el objeto `PEAR::MDB2` o `PDO` conectado

```
/*CONEXION ACTIVA DE LA CLASE MANEJADORA*/
//Recogemos el objeto IgepConexion de la conexion activa.
$conIgepConexion = $this->getConnection(); // $this es la clase manejadora
//Recogemos el objeto PEAR::MDB2 de la conexion activa.
$conPEARMDB2 = $conIgepConexion->getConnection();
//Recogemos el driver nativo de PHP de la conexion activa sólo en el caso PEAR::MDB2
$conNativePHP = $conPEARMDB2->getConnection();
```

```
/* NUEVA CONEXIÓN */
$conIgepConexion = new IgepConexion($g_dns_ora);
//Recogemos el objeto PEAR::MDB2
$conPEARMDB2 = $conIgepConexion->getConnection();
//Recogemos el driver nativo de PHP
$conNativePHP = $conPEARMDB2->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 [161] 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 );//ATENCIÓN 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 nullos

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. gvHidraDBAL

GVHidraDBAL es una clase incorporada como utilidad en las últimas versiones de GVHidra. Su finalidad es facilitar un driver sencillo para el programador, que le permita trabajar como el nivel mas bajo de la clase IgepConexion, pero de forma independiente. La idea es que esta clase (o alguna evolución de la misma) pase a ser utilizada por IgepConexión y sustituya a IgepDBMS.

Como principal novedad, GVHidraDBAL ofrece compatibilidad con los DNSs de gvHidra (y por tanto de MDB2). La clase se implementa con una factoria, que permite instanciar objetos con la misma interfaz, pero particularidades propias en cuanto al driver o forma de comunicarse con el SGBD. La versión actual de gvHidraDBAL, permite:

- MDB2 (PostgreSQL, MySQL, Oracle)
- PDO (PostgreSQL, MySQL, Oracle, SQLite)
- NATIVE (Oracle)

Pudiendo así de forma optar por la mejor solución (por ejemplo en base a las prestaciones) en cada caso.

Además de lo expuesto:

- Añade algunos métodos de consulta que no estaban disponibles en IgepConexión (por ejemplo, la obtención mediante fetch() de registros en modo cursor (de uno en uno), para evitar consumir la memoria si el resultado de una consulta es muy numeroso en registros.
- No utiliza internamente la clase IgepTransformer para convertir fechas y formatos numéricos desde el SGBD al PHP (y viceversa) sino que se configura (junto con otros parámetros como la codificación) dichas opciones y a partir de ahí es transparente para el programador (que suele conocer el formato admitido por el SGBD concreto y el que quiere en su aplicación).

Por último comentar, que la herramienta IgepDebug (tanto la clase que realiza las inserciones como el visor), utilizan gvHidraDBAL de forma interna. A continuación podemos ver algunos ejemplos de uso:

```
!!!!PENDIENTE!!!!
```

5.2.1. API de gvHidraDBAL

5.2.1.1. gvHidraDBAL::gvHidraDBAL class Reference

5.2.1.1.1. Campos de datos

- const DEBUG
- const DEFAULT_CLIENT_ENCODING
- const DEFAULT_CONNECTION_TYPE
- const PDO_DRIVER

- const MDB2_DRIVER
- const NATIVE_DRIVER
- const SGBD_POSTGRESQL
- const SGBD_ORACLE
- const SGBD_MYSQL
- const SGBD_MARIADB
- const SGBD_SQLITE
- const T_INTEGER
- const T_DECIMAL
- const T_STRING
- const T_CHAR
- const T_BOOLEAN
- const T_DATE
- const T_DATETIME
- const T_TIMESTAMP
- const T_BLOB
- const T_CLOB

5.2.1.1.2.

- static \$SGBD_dateTimeFormat
- static \$SGBD_numericFormat
- static \$v_oracle
- static \$v_postgres
- static \$v_mysql
- static \$v_sqlite
- static \$v_latin1
- static \$v_latin9
- static \$v_utf8

5.2.1.1.3. Atributos protegidos

- \$forcedDateTimeFormat
- \$phpDateMask
- \$phpTimeMask

- \$phpDatetimeMask
- \$phpTimestampMask
- \$phpDatetimeMaskTZ
- \$phpTimestampMaskTZ
- \$SGBDDateMask
- \$SGBDTimeMask
- \$SGBDDatetimeMask
- \$SGBDTimestampMask
- \$SGBDDatetimeMaskTZ
- \$SGBDTimestampMaskTZ
- \$forcedNumericFormat
- \$decimalSeparator
- \$thousandSeparator
- \$clientEncoding
- \$connectionType
- \$tipoSQL
- \$SGBDType
- \$conDB
- \$statement
- \$resultSet
- \$affectedRows
- \$vTypesManipulationParams
- \$vTypesResultValues

5.2.1.1.4.

- setQueryType (\$query, \$vTypesResultValues)
- inicializacionGenerica (\$dsn, \$encoding)

5.2.1.1.5.

- static SGBDAllowForceDatetimeFormat (\$dsn)
- static SGBDMask (\$dsn, \$tipo, \$formato)
- static SGBDForcedNumericFormat (\$dsn)
- static getSGBDDecimalSeparator (\$dsn)

- static getSGBDThousandSeparator (\$dsn)
- getNativeThousandSeparator ()
- static getNormalizedSGBDType (\$dsn)
- static makeConnection (\$dsn, \$encoding, \$connectionType)

5.2.1.1.6.

- isForcedDateTimeFormat ()
- getMask (\$tipo, \$formato)
- setMask (\$tipo, \$maskPHP, \$maskSGBD)
- isForcedNumericFormat ()
- forceNumericFormat (\$decimalSeparator, \$thousandSeparator)
- getNativeDecimalSeparator ()
- prepare (\$query, \$vTypesManipulationParams, \$vTypesResultValues)
- execute (\$vValuesBindParams)
- fetch ()
- fetchAll ()
- close (\$force)
- beginTransaction ()
- commit ()
- rollback ()
- inTransaction ()
- fetchAll4Query (\$sqlQuery)
- execute4Query (\$sqlInsertUpdate)
- acabarTransaccion (\$error)
- consultar (\$sqlQuery)
- operar (\$sqlInsertUpdate)
- isLOB (\$fieldLob)
- LOBAsString (\$fieldLob)
- LOBAsStream (\$fieldLob)
- transformToPHP (\$fieldValue, \$tipo)
- transformToSGBD (\$fieldValue, \$tipo)

5.2.1.1.7.

- static isAssoc (\$arr)

5.2.1.1.8.

- static makeConnectionNative (\$dsn, \$encoding)

5.2.1.1.9. Documentación de los campos

5.2.1.1.10. Documentación de los campos

5.2.1.1.11. setQueryType (\$query, \$vTypesResultValues)

Determina el tipo de query que va a ejecutarse.

protected

Tabla 5.2. Parámetros

\$query	Cadena SQL (SQL, insert, update / delete).
\$vTypesResult	Values vector (asociativo o en orden) con los tipos de las columnas devueltas por la operación (si es una select). NULL si no se devuelven datos.

Tabla 5.3. Excepciones

Exception	
-----------	--

5.2.1.1.12. inicializacionGenerica (\$dsn, \$encoding=self::DEFAULT_CLIENT_ENCODING)

inicializacionGenerica. Recibe un dsn gvHidra para conectar y opcionalmente la codificación.

protected

Tabla 5.4. Parámetros

\$dsn	Array asociativo con los datos de conexión. Basado en el dns propuesto en MDB2.
\$encoding	Codificación.

Tabla 5.5. Excepciones

Exception	
-----------	--

Devuelve: . array Array asociativo con la cadena de conexión PDO necesaria o los parámetros MDB2.

5.2.1.1.13. static SGBDAllowForceDatetimeFormat (\$dsn)

Devuelve si el SGBD permite sobrescribir formatos de fechas.

Tabla 5.6. Parámetros

\$dsn	Array asociativo con los datos de conexión.
-------	---

\$tipo	[date time datetime timestamp]
--------	--------------------------------------

Devuelve: . string

5.2.1.1.14. static SGBDMask (\$dsn, \$tipo, \$formato)

Devuelve la máscara correspondiente a \$tipo, en sintaxis PHP o la nativa del SGBD especificado en el DNS.

Tabla 5.7. Parámetros

\$dsn	Array asociativo con los datos de conexión
\$tipo	[date time datetime timestamp]
\$formato	[php sgbd]

Devuelve: . string

5.2.1.1.15. static SGBDForcedNumericFormat (\$dsn)

Devuelve null o si el SGBD permite forzar formato numérico, el equivalente al de PHP en sintaxis SGBD.

Tabla 5.8. Parámetros

\$dsn	Array asociativo con los datos de conexión
-------	--

Devuelve: . string

5.2.1.1.16. static getSGBDDecimalSeparator (\$dsn)

Devuelve el separador decimal fijado por defecto para ese SGBD

Tabla 5.9. Parámetros

\$dsn	Array asociativo con los datos de conexión
-------	--

Devuelve: . string Carácter/cadena.

5.2.1.1.17. static getSGBDThousandSeparator (\$dsn)

Devuelve el separador de miles fijado por defecto para ese SGBD.

Devuelve: . string Carácter/cadena.

5.2.1.1.18. getNativeThousandSeparator ()

Devuelve el separador de miles utilizado en esta conexión para el manejo de números reales.

Devuelve: . string El caracter o cadena definida como separador decimal.

5.2.1.1.19. static getNormalizedSGBDType (\$dsn)

Obtiene el alias del tipo de conexión según el SGBD (oci, pgsq, mysql, sqlite...).

Tabla 5.10. Parámetros

\$dsn	Array asociativo con los datos de conexión.
-------	---

Tabla 5.11. Excepciones

Exception	
-----------	--

Devuelve: . string

5.2.1.1.20. static makeConnection (\$dsn, \$encoding=self::DEFAULT_CLIENT_ENCODING, \$connectionType=self::DEFAULT_CONNECTION_TYPE)

makeConnection. Devuelve una instancia de la clase gvHidraDBAL correspondiente a los parametros de entrada.

Tabla 5.12. Parámetros

\$dsn	Array asociativo con los datos de conexión. Basado en el dns propuesto en MDB2.
\$encoding	Codificación
\$connectionType	Driver a utilizar en la conexión a BD [MDB2, PDO, NATIVE]

Tabla 5.13. Excepciones

Exception	
-----------	--

Devuelve: . gvHidraDBAL Instancia de la clase gvHidraDBAL o heredera.

5.2.1.1.21. isForcedDateTimeFormat ()

Devuelve si en esa conexión puede forzarse un formato para fechas y horas.

Devuelve: . boolean

5.2.1.1.22. getMask (\$tipo, \$formato='php')

Devuelve la máscara correspondiente a \$tipo, en sintáxis PHP para la conexión actual.

Tabla 5.14. Parámetros

\$tipo	[date time datetime timestamp]
\$formato	[php sgbd]

Devuelve: . string

5.2.1.1.23. setMask (\$tipo, \$maskPHP, \$maskSGBD)

Fija las máscara correspondiente a \$tipo, en sintáxis PHP y SGBD para la conexión actual si se permite sobrescritura.

Tabla 5.15. Parámetros

\$tipo	[date time datetime timestamp]
\$maskPHP	Mascara en formato PHP http://php.net/manual/en/function.date.php
\$maskSGBD	Mascara en la sintaxis propia del SGBD (si la tiene).

Devuelve: . string

5.2.1.1.24. isForcedNumericFormat ()

Devuelve si en esa conexión puede forzarse un formato para reales.

Devuelve: . boolean

5.2.1.1.25. forceNumericFormat (\$decimalSeparator, \$thousandSeparator)

Fija el formato fijado esperado por el SGBD para manejo de números en esta conexión si es posible sobrecargarlo, en otro caso, lo ignora.

Tabla 5.16. Parámetros

\$decimalSeparator	
\$thousandSeparator	

Devuelve: . string Cadena con formato PHP <http://php.net/manual/en/function.date.php>.

5.2.1.1.26. getNativeDecimalSeparator ()

Devuelve el separador decimal utilizado en esta conexión para el manejo de números reales.

Devuelve: . string El caracter o cadena definida como separador decimal.

5.2.1.1.27. prepare (\$query, \$vTypesManipulationParams=null, \$vTypesResultValues=null)

Prepara para su ejecución una sentencia SQL con parámetros (si procede). Este método debe implementarse en las clase heredadas.

Ejemplo1 \$query 'SELECT user as "usuario", height as "altura" from t_usuarios where cod_pro=:codProvincia' \$vTypesManipulationParams ['codProvincia' => gvHidraDBAL::STRING]

Ejemplo2 \$query = 'UPDATE t_usuarios SET weight =? where f_nacimiento >=?'

NOTA: Debido a una limitación de MDB2 en el uso de lobs (clob y blob). Si se van a realizar operaciones de inserción/actualización con estos campos, debemos hacer uso de todos los parámetros.

Ejemplo:

Tabla 5.17. Parámetros

\$query	Cadena SQL (SQL, insert, uptate / delete).
\$vTypesManipulationParams	Vector (asociativo o en orden) con los tipos de las variables de manipulación (si existen), serían los tipos de los parámetros WHERE o de los valores en INSERT o UPDATES.

\$vTypesResultValues	Vector (asociativo o en orden) con los tipos de las columnas devueltas por la operación (si es una select). NULL si no se devuelven datos (insert/update/upsert...).
----------------------	--

Tabla 5.18. Excepciones

Exception	
-----------	--

5.2.1.1.28. execute (\$vValuesBindParams=null)

Ejecuta la sentencia SQL previamente preparada Este método debe implementarse en las clase herederas

Ejemplo 1 (asociativo con respecto a las variables :VARBIND) IN ([:usuario => 'pascual_dav', :altura => 185])

Ejemplo 2 (no asociativo, pero ordenado con respecto a las variables ?) IN (['pascual_dav', 185])

Tabla 5.19. Parámetros

\$vValuesBindParams	Vector (asociativo o ordenado) con los valores de las variables (si existen) de la query.
---------------------	---

Tabla 5.20. Excepciones

Exception	
-----------	--

5.2.1.1.29. fetch ()

Obtiene una fila de un conjunto de resultados asociado al Statement. Este método debe implementarse en las clase herederas.

Tabla 5.21. Excepciones

Exception	
-----------	--

Devuelve: . array|bool Devuelve el registro siguiente (array asociativo) o false si error.

5.2.1.1.30. fetchAll ()

Devuelve una matriz de arrays asociativos equivalentes a cada uno de los registros resultado de la consulta, o array vacío si no hay datos. Este método debe implementarse en las clase herederas.

Tabla 5.22. Excepciones

Exception	
-----------	--

Devuelve: . array Matriz de vectores asociativos.

5.2.1.1.31. close (\$force=false)

Libera la conexión de la base de datos. Este método debe implementarse en las clase herederas.

Tabla 5.23. Parámetros

\$force	Indica si se intenta forzar el cierre.
---------	--

Tabla 5.24. Excepciones

Exception	
-----------	--

5.2.1.1.32. beginTransaction ()

Inicia la transacción. Este método debe implementarse en las clase herederas.

Tabla 5.25. Excepciones

Exception	
-----------	--

Devuelve: . mixed Devuelve el resultado.

5.2.1.1.33. commit ()

Finaliza la transacción con commit. Este método debe implementarse en las clase herederas.

Devuelve: . mixed Devuelve el resultado.

5.2.1.1.34. rollback ()

Finaliza la transacción con rollback. Este método debe implementarse en las clase herederas.

Devuelve: . mixed Devuelve el resultado.

5.2.1.1.35. inTransaction ()

Indica si hay una transacción en proceso. Este método debe implementarse en las clase herederas.

Devuelve: . int|bool Indica si hay una transaccion en proceso (MDB2 => Entero: N° de transacciones anidadas, Bool: Si hay o no hay transaccion).

5.2.1.1.36. fetchAll4Query (\$sqlQuery)

Devuelve una matriz de arrays asociativos equivalentes a cada uno de los registros resultado de la consulta, o array vacío si no hay datos. Este método debe implementarse en las clase herederas.

Tabla 5.26. Parámetros

\$sqlQuery	Sentencia SELECT a realizar.
------------	------------------------------

Tabla 5.27. Excepciones

Exception	
-----------	--

Devuelve: . array Matriz de arrays asociativos.

5.2.1.1.37. execute4Query (\$sqlInsertUpdate)

Ejecuta la sentencia SQL suministrada de tipo INSERT o UPDATE. Este método debe implementarse en las clase herederas.

Tabla 5.28. Parámetros

\$sqlInsertUpdate	Sentencia a realizar (INSERT / UPDATE).
-------------------	---

Tabla 5.29. Excepciones

Exception	
-----------	--

Devuelve: . int|bool Devuelve el numero de filas afectadas, o FALSE si ha ocurrido un error.

5.2.1.1.38. acabarTransaccion (\$error)

Finaliza la transacción. Realiza un rollback si se indica error, o un commit si no hay error.

Tabla 5.30. Parámetros

\$error	Indica si hay que error, o se puede completar la transaccion.
---------	---

Devuelve: . mixed Devuelve el resultado.

5.2.1.1.39. consultar (\$sqlQuery)

Devuelve una matriz de arrays asociativos equivalentes a cada uno de los registros resultado de la consulta, o array vacío si no hay datos (Alias de fetchAll4Query)

Tabla 5.31. Parámetros

\$sqlQuery	Sentencia SELECT a realizar.
------------	------------------------------

Devuelve: . array Matriz de arrays asociativos.

5.2.1.1.40. operar (\$sqlInsertUpdate)

Ejecuta la sentencia SQL suministrada de tipo INSERT o UPDATE. (Alias de execute4Query)

Tabla 5.32. Parámetros

\$sqlInsertUpdate	Sentencia a realizar (INSERT / UPDATE).
-------------------	---

Devuelve: . int|bool Devuelve el numero de filas afectadas, o FALSE si ha ocurrido un error.

5.2.1.1.41. isLOB (\$fieldLob)

Determina si valor suministrado contiene un LOB(BLOB / CLOB).

Tabla 5.33. Parámetros

\$fieldLob	LOB del que devolver el contenido.
------------	------------------------------------

Devuelve: . bool Indica si ha suministrado un LOB.

5.2.1.1.42. LOBAsString (\$fieldLob)

Devuelve el contenido de un campo de tipo LOB (BLOB / CLOB).

Tabla 5.34. Parámetros

\$fieldLob	LOB del que devolver el contenido.
------------	------------------------------------

Devuelve: . string Contenido del LOB.

5.2.1.1.43. LOBAsStream (\$fieldLob)

Devuelve un stream con el contenido de un campo de tipo LOB (BLOB / CLOB).

Tabla 5.35. Parámetros

\$fieldLob	LOB del que devolver el contenido.
------------	------------------------------------

Devuelve: . string Stream que contiene LOB.

5.2.1.1.44. transformToPHP (\$fieldValue, \$tipo)

Transforma una fecha de representacion en SGBD a su representacion PHP (Datetime) usando el tipo especificado.

Tabla 5.36. Parámetros

\$fieldValue	Valor a transformar.
\$tipo	[date time datetime timestamp].

Devuelve: . Valor transformado.

5.2.1.1.45. transformToSGBD (\$fieldValue, \$tipo)

Transforma una fecha (Datetime) de representacion en PHP a su representacion SGBD usando el tipo especificado.

Tabla 5.37. Parámetros

\$fieldValue	Valor a transformar.
\$tipo	[date time datetime timestamp].

Devuelve: . string Valor transformado.

5.2.1.1.46. static isAssoc (\$arr)

Determina si un array es asociativo o numérico (con saltos o no).

protected

Tabla 5.38. Parámetros

\$arr	
-------	--

5.2.1.1.47. static makeConnectionNative (\$dsn, \$encoding=self::DEFAULT_CLIENT_ENCODING)

makeConnection. Devuelve una instancia de una clase gvHidraDBAL nativa correspondiente a los parametros de entrada.

private

Tabla 5.39. Parámetros

\$dsn	Array asociativo con los datos de conexión. Basado en el dns propuesto en MDB2.
\$encoding	Codificación.
\$connectionType	Driver a utilizar en la conexion a BD.

Tabla 5.40. Excepciones

Exception	
-----------	--

Devuelve: . gvHidraDBAL Instancia nativa de la clase gvHidraDBAL.

5.3. Web Services

Un Servicio Web (Web Service), es un componente lejano y externo al sistema, que proporciona funcionalidad a través de protocolos Web estándar, de forma independiente a la tecnología utilizada.

Dentro de los servicios hay multitud de propuestas, destacando, los dos estándar siguientes:

- *SOAP*, basado en mensajes XML con una estructura prefijada, donde un *sobre (Envelope)* describe el mensaje, a quien va dirigido, y cómo debe ser procesado. Se estructura en:
 - *Cuerpo (Header)* opcional. Se puede incluir información sobre el mensaje. Por ejemplo, el estándar WS-Security utiliza las cabeceras para añadir seguridad (autenticación, cifrados...) extendiendo SOAP.
 - *Cuerpo (Body)*, que contiene el mensaje en si. Puede (opcionalmente) contener información sobre posibles errores.
 - Error (Fault) que indicar en una respuesta SOAP errores en el procesamiento del mensaje.
- *REST* o RESTful (Representational State Transfer). Es una alternativa más sencilla que SOAP (también menos potente). Se estructura directamente sobre el protocolo HTTP (PUT, GET, POST,DELETE). No tiene estado y basa el intercambio de datos en XML y/o JSON

5.3.1. Web Services en PHP

PHP cuenta, desde las versión PHP 5.0, con soporte nativo (propio del lenguaje) para el consumo de servicios Web SOAP v1.1 y SOAP v1.2. Sin embargo, PHP no tiene soporte nativo para muchas de sus extensiones, como (entre otras) WS-Security o tratamiento de MTOM (anexos binarios del mensaje).

5.3.1.1. 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 (archivo 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
```

5.3.1.2. 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 archivo php (en nuestro ejemplo server.php) que contendrá las llamadas a las clases SOAP correspondientes al servidor. En este mismo archivo 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 archivo 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');
$server->service($_SERVER['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 archivo WSDL. El archivo 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.

5.3.2. Web Services en gvHIDRA. WSComun

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.

Capítulo 6. Seguridad

6.1. Autenticación de usuarios

6.1.1. Introducción

gvHIDRA permite integrar las aplicaciones con diferentes sistemas de autenticación 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 núcleo por lo que ya no se necesitará tener instalado en nuestro servidor el paquete Auth. Para comprender el funcionamiento, conviene distinguir dos partes:

1. Autenticación: validar que el usuario es quien dice ser. El framework proporciona el método fetchData para realizar dicha comprobación.
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`.

A continuación veremos algunos de los mecanismos ya implementados y como se tratan estos puntos en cada uno de ellos.

En el paquete que se distribuye del framework se puede encontrar ejemplos de validación en el directorio "samples", validación usuario/contraseña como validación con captcha.

6.1.1.1. AuthBasic

Es el método usado por defecto por el framework. En este sistema en el método de autenticación valida que las credenciales son **invitado-1**. En el método de autorización carga una configuración básica.

Para hacer uso de este método hay que copiar el fichero `igep/include/valida/login.php` a la raíz del proyecto, y acceder a él con el navegador.



6.1.1.2. AuthCaptcha

A la validación básica de usuario/contraseña se puede completar con un captcha. Captcha es una medida de seguridad conocida como autenticación pregunta-respuesta, que ayuda a protegerse del spam y del descifrado de contraseñas pidiendo que completes una simple prueba que demuestre que eres humano y no un ordenador que intenta acceder a una cuenta protegida con contraseña.

Para ello, en el fichero de entrada `login.php`, se debe incluir el fichero `AuthCaptcha.php`



6.1.2. Crear un nuevo método de Autenticación

A continuación se explican los pasos para crear el nuevo método.

Excepto el fichero de índice (paso 1), los otros pueden situarse en cualquier ubicación, y normalmente vendrá determinado en función de donde necesitemos usarlos (aplicación, custom o framework). En aplicación, por ejemplo, una ubicación habitual es en `app/include/valida`.

6.1.2.1. Validación usuario/contraseña

6.1.2.1.1. 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

```
function fetchData($username, $password, $isChallengeResponse=false) {
    // Check If valid etc
    $usuarios = array('consultor','tramitador','administrador','informatico');

    if(in_array($username, $usuarios)) {
        return true;
    }
    return false;
}
```

- `authenticate`: 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 3, en el ejemplo `"include/valida/validacion.php"`.

```
static function authenticate($p_apli) {
    $auth_container = new self($p_apli);

    IgepSession::session_start($p_apli, false);
    $aut = new Auth($auth_container, '', 'loginFunction');
    $aut->setSessionName($p_apli);
}
```

```
$aut->setAuthData('lang', $_REQUEST['language']);

if (isset($_GET['logout']))
    $aut->logout();
$aut->start();

$resp = $auth_container->getResponse();

if ($aut->checkAuth()) {
    $aut->setAuthData('response', $resp);
    $auth_container->open('include/valida/validacion.php');
    return '';
} else {
    if (isset($resp))
        return $resp->getMessage();
    else
        return ''; // ocurre cuando no hay usuario/password
}
}
```

6.1.2.1.2. Paso 2. Fichero de entrada a la aplicación: login.php

Creamos el fichero que invoca al método autenticate anterior. Este fichero siempre ha de ir en la raíz del proyecto, y es el punto de entrada a la aplicación usando el navegador.

```
include_once('igep/include_class.php');
include_once 'include/valida/AuthBasic.php';
$msg = AuthBasic::autenticate(ConfigFramework::getApplicationName());
if ($msg)
{
    echo $msg;
}
```

6.1.2.1.3. Paso 3. Clase de validación

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.

```
require_once 'AuthBasic.php';
class validacion
{
    static function valida($apli, $sesion=TRUE)
    {
        $auth_container = new AuthBasic();
        $aut = new Auth($auth_container, '', '', false);
        $aut->setSessionName($apli);
        $aut->start();
        if (!$aut->checkAuth())
            exit('No estás validado; Vuelve a la pantalla de conexión.');
```

```

    if (!isset($_SESSION[$apli]['usuario'])) {
        if (!isset($_SESSION[$apli]))
            $_SESSION[$apli] = array();
        $_SESSION[$apli]['usuario']['usuario'] = $aut->getUsername();
        $_SESSION[$apli] = $auth_container->postLogin($_SESSION[$apli], $aut);
        $_SESSION['validacion'] = $auth_container->getDatos();
    }
}
```



```

    $auth_container->checkData($_SESSION, $apli);
}
}
}

```

6.1.2.2. Validación usuario/contraseña con captcha

6.1.2.2.1. 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/gvhBaseCaptcha.php'. Los métodos que debemos implementar son:

- `fetchData`: primero realiza la comprobación del captcha y después comprueba la validez del usuario y la contraseña.
- `authenticate`: es el método llamado desde el paso el fichero principal `login.php`, y si se aceptan las credenciales, llamamos al método `open` pasándole la URL de la clase que definimos en paso 3, en el ejemplo "include/valida/validacion.php".

6.1.2.2.2. Paso 2. Fichero de entrada a la aplicación: login.php

Este fichero es igual que el que creamos si utilizamos validación usuario/contraseña, pero incluyendo el fichero `AuthCaptcha.php`.

```
include_once 'include/valida/AuthCaptcha.php';
```

6.1.2.2.3. Paso 3. Clase de validación

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`.

```

require_once 'AuthCaptcha.php';
class validacion
{
    public static function valida($apli, $sesion=TRUE)
    {
        $auth_container = new AuthCaptcha();
        $aut = new Auth($auth_container, '', '', false);
        $aut->setSessionName($apli);
        $aut->start();
        if (!$aut->checkAuth())
            exit('No estás validado: Vuelve a la pantalla de conexión.');
```

```

        $entorno = trim(strtoupper(ConfigFramework::getConfig()-
>getProperty('p_entorno')));
        if (!isset($_SESSION[$apli]['usuario']))
        {
            if (!isset($_SESSION[$apli]))
            {
                $_SESSION[$apli] = array();
            }

            $_SESSION[$apli]['usuario']['usuario'] = $aut->getUsername();
            $_SESSION[$apli] = $auth_container->postLogin($_SESSION[$apli], $aut);
            $_SESSION['validacion'] = array(
                'bd'=>$entorno,
                'server'=>'host-login',
            );

```

```
$auth_container->checkData($_SESSION, $apli);
}
}
}
```

6.2. Módulos 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, sería 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 sería 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 sería 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 desactivación 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.

Parte IV. Conceptos Avanzados

Tabla de contenidos

7. Conceptos Avanzados	161
7.1. Excepciones	161
7.1.1. gvHidraSQLException	161
7.1.2. gvHidraLockException	161
7.1.3. gvHidraPrepareException	161
7.1.4. gvHidraExecuteException	161
7.1.5. gvHidraFetchException	161
7.1.6. gvHidraNotInTransException	162
7.2. Log de Eventos	162
7.2.1. Introducción	162
7.2.2. Crear eventos en el log	163
7.2.3. Consulta del Log	163
7.3. Herramientas para debug de la aplicación.	164
7.3.1. xDebug	164
7.3.2. Consola del navegador (Firefox)	171
7.3.3. Debug de plantillas Smarty	173
7.4. CUSTOM	174
7.4.1. Pasos previos	174
7.4.2. Archivos CSS	174
7.4.3. Clases más usadas de bootstrap	175
7.4.4. Iconos mediante fuentes tipográficas (icon-fonts)	175
7.4.5. Tipos de clases	176
7.4.6. Cosas a tener en cuenta	176
7.4.7. Estilos generales de la aplicación	176
7.4.8. Personalización de Logos	177
7.4.9. Panel login	179
7.4.10. Pantalla de inicio	182
7.4.11. Acerca de	186
7.4.12. Ventanas de aviso	188
7.4.13. Partes genéricas de los paneles FIL LIS EDI	190
7.4.14. Partes genéricas de los paneles LIS EDI	196
7.4.15. Panel FIL	199
7.4.16. Panel EDI	201
7.4.17. Panel LIS	202
7.4.18. Panel maestro-detalle	205
7.4.19. Panel árbol	207
7.4.20. Solapas	208
7.4.21. Debugger	210
7.4.22. Ejemplos prácticos	211

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 jerarquía 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 puede 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 debería seguir ejecutándose
ERROR	Errores

tipo	descripción
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 línea:

```
<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. Herramientas para debug de la aplicación.

Además del propio debugger de la aplicación interesa instalarse alguna herramienta que nos ayude a depurar con más precisión la programación de nuestra aplicación. Vamos a explicar dos opciones que serán de utilidad.

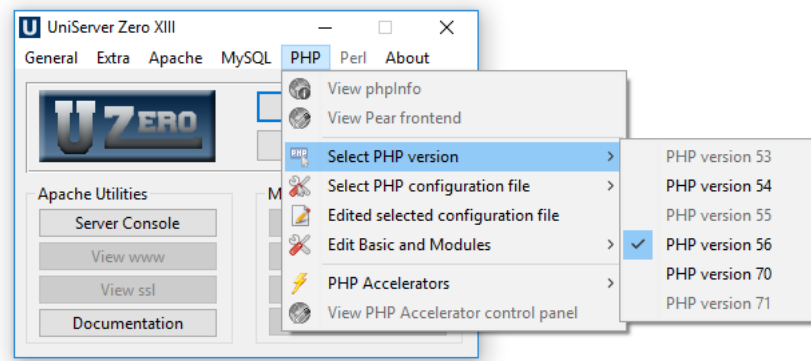
7.3.1. xDebug

Es una extensión de PHP para hacer debug con herramientas de depuración tradicionales, desde el editor, tal como se hace en lenguajes de programación clásicos. Con xDebug se permite realizar el seguimiento del flujo de ejecución e ir analizando el contenido de las variables en cada momento. Ofreciendo también la posibilidad de marcar puntos de ruptura en el código donde el flujo del programa se detendrá y así poder ver el estado de las variables, o la traza de ejecución.

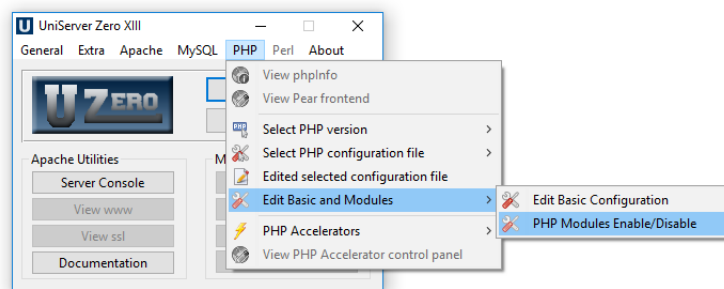
7.3.1.1. Configuración en PortableApps de gvHIDRA

7.3.1.1.1. PHP - Ficheros ".ini"

Abrimos **UniServer Zero** del Portable Apps de gvHidra y sin arrancar el servicio de Apache (por tanto, si está iniciado el servicio Apache, deberemos detenerlo antes de continuar), y seleccionamos la versión PHP que vamos a configurar, en este caso la versión PHP 5.6, a través del menú **PHP > Select PHP version > PHP version 5.6**



Abrimos la edición de módulos y comprobamos que la opción “**php_xdebug.dll**” está marcada, a través del siguiente menú de Uniserver Zero:



Abrimos la ruta de los ficheros de configuración de PHP, en el caso del PortableApps se encontrará en #ruta instalación portable#\PortableApps\UniServerZ\core\php56. Abriremos los ficheros “**php_development.ini**”, “**php_production.ini**”, “**php_test.ini**” y “**php-cli.ini**”.

En el fichero “**php_development.ini**” buscamos el bloque de configuración de Xdebug, y eliminamos el punto y coma de la primera línea. Comprobamos que la ruta asignada es la que corresponde con la versión PHP 5.6.

```
[xdebug]
zend_extension=${US_ROOTF}/core/php56/extensions/php_xdebug.dll
```

```
xdebug.remote_autostart=off
xdebug.remote_enable=on
xdebug.remote_host=127.0.0.1
xdebug.remote_port=9000
xdebug.remote_handler=dbgp
xdebug.remote_mode=req
xdebug.idekey=default
```

Dichas líneas de código también las utilizaremos lo sustituimos en los demás ficheros ‘ini’ (“**php_production.ini**”, “**php_test.ini**” y “**php-cli.ini**”) para que, cuando seleccionemos cualquier otro entorno de trabajo, también tengamos disponible la depuración mediante de xdebug.

Por último, arrancamos el servicio de apache y abrimos el fichero “**phpinfo.php**” en un navegador, que en el caso del PortableApps de gvHidra se encuentra en la dirección http://localhost/us_extra/phpinfo.php. Comprobamos que aparece el logo de Zend Engine en el pie de la primera tabla (el logo puede diferir según la versión de PHP activa).

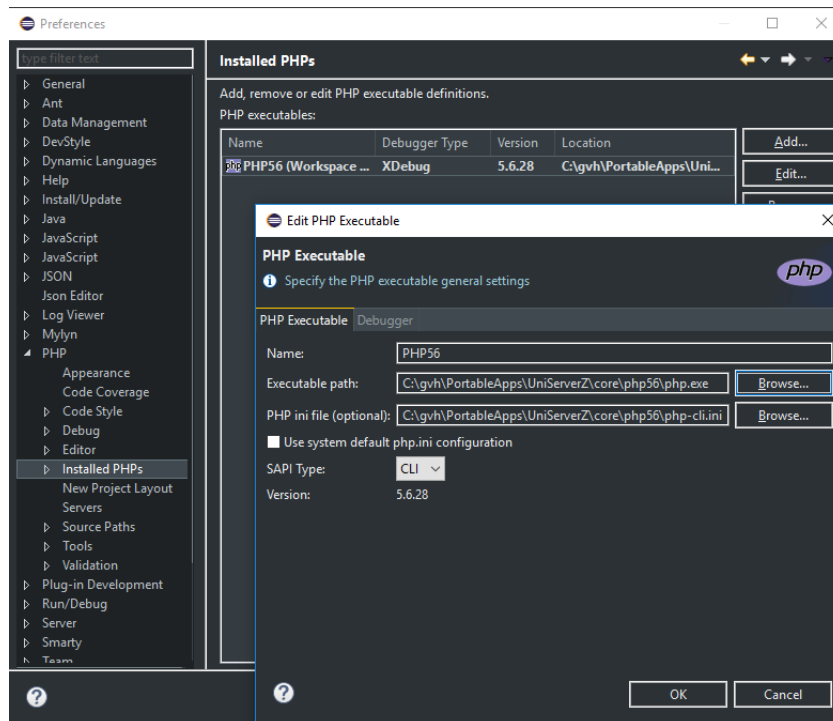


Vamos a la tabla de Xdebug y comprobamos que la configuración del host y del puerto es la correcta, y que se corresponden con las líneas de configuración que habíamos añadido a los ficheros ‘ini’.

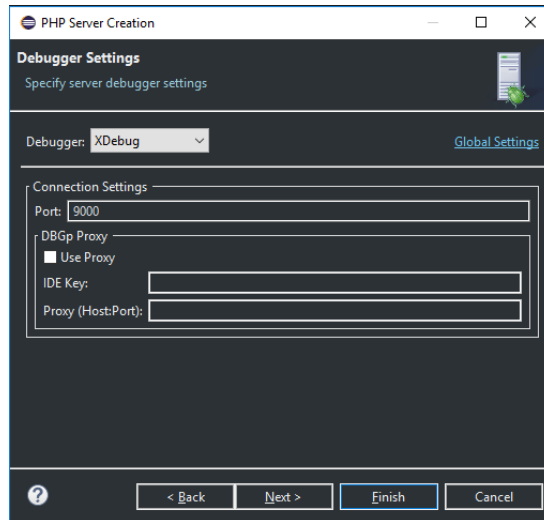
xdebug.remote_host	127.0.0.1	127.0.0.1
xdebug.remote_log	no value	no value
xdebug.remote_mode	req	req
xdebug.remote_port	9000	9000

7.3.1.1.2. Eclipse

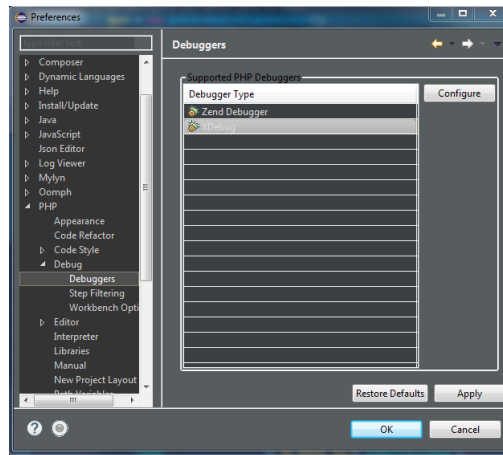
Abrimos el menú **Window > Preferences > PHP > Installed PHPs** y añadimos una nueva configuración. Insertamos la ruta del ejecutable de PHP en *Executable path*, y la ruta del "**php-cli.ini**" en el campo PHP ini file.



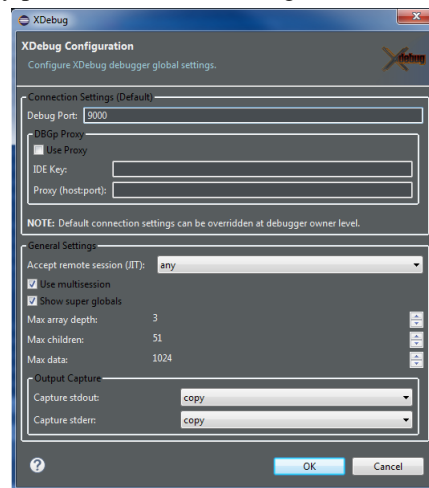
En la pestaña Debugger le ponemos el puerto 9000, es decir, el mismo número de puerto que habíamos añadido al fichero de configuración "ini" de PHP. Abrimos la pestaña **PHP > Servers** y añadimos un nuevo servidor. Le ponemos un nombre y le asignamos la url base <http://localhost>. En la siguiente pestaña seleccionamos el debugger **Xdebug** y le asignamos el puerto 9000. Por último seleccionamos finish.



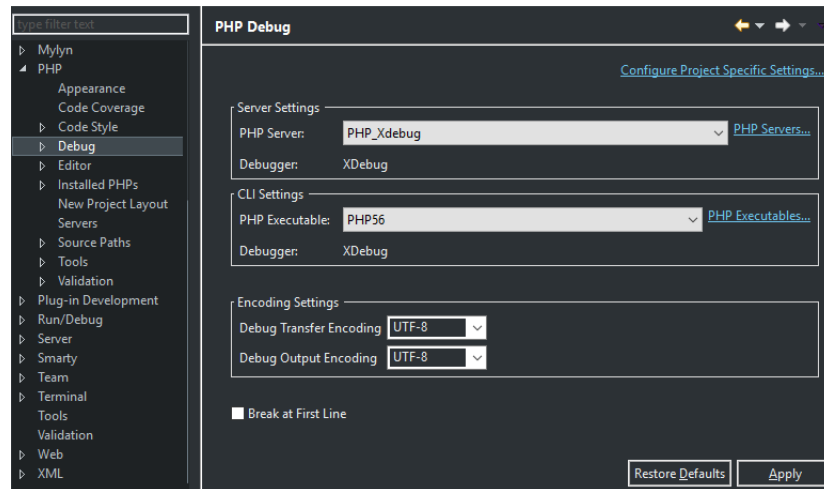
Dentro de la opción **Debug > Debuggers**.



Seleccionamos de la lista el Xdebug y pulsamos el botón **Configure**.

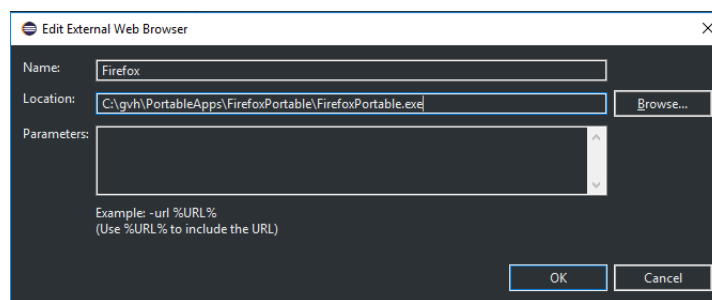


Debemos seleccionar dentro de **General Settings** en la opción **Accept remote session (JIT)** el valor **any** y confirmar los cambios con **OK**. Una vez añadida la configuración de PHP y el servidor Xdebug abrimos la pestaña **PHP > Debug**. En ella seleccionamos el servidor que acabamos de crear, y la configuración del ejecutable. Aplicamos los cambios y cerramos

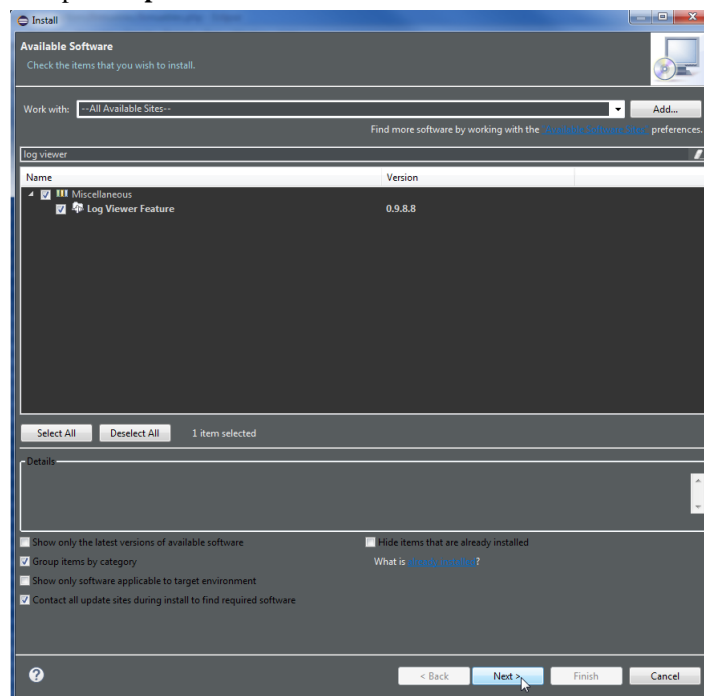


Por último configuraremos el navegador por defecto, en nuestro caso Mozilla Firefox que hay disponible en el propio PortableApps. Para ello abriremos la pestaña **General > Web Browser**. Seleccionamos **Use external web browser** y añadiremos un nuevo navegador a la lista.

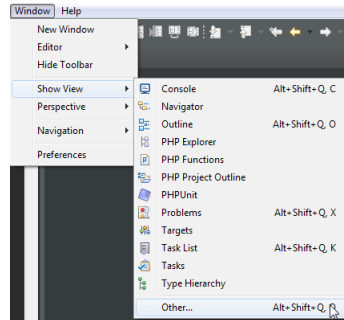
En el campo **Location** ponemos la ruta del ejecutable del navegador, en este caso la ruta del ejecutable **FirefoxPortable.exe**.



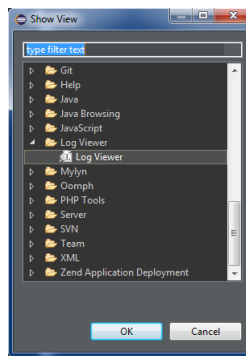
De forma opcional, si se quiere ver el log de apache integrado en Eclipse se puede instalar el complemento Log Viewer desde el menú de eclipse **Help > Install new software**



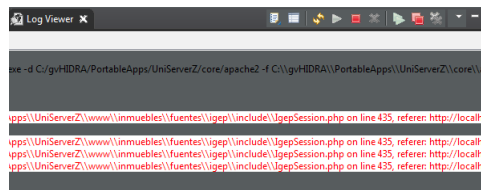
Una vez instalado vamos a **Window > Show View > Other.**



Seleccionamos la vista *Log Viewer*



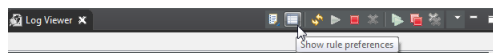
Al darle a OK nos aparecerá una nueva pestaña en la parte inferior de eclipse.



Para abrir el fichero de log que queremos visualizar pulsamos el botón Open Logfile y seleccionamos la ruta del archivo.

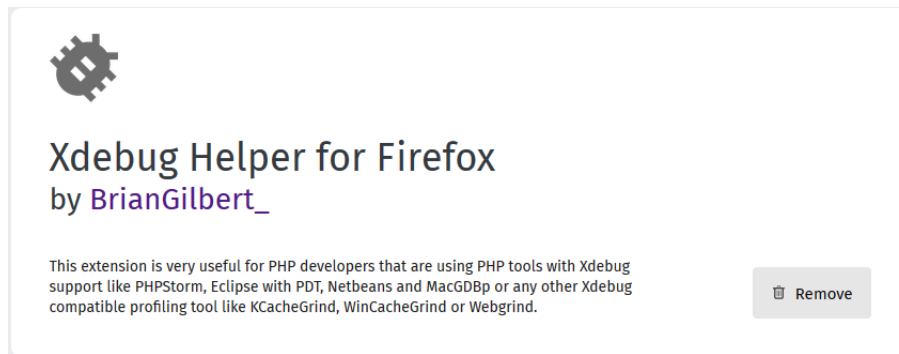


Si se desea aplicar reglas de coloreado al fichero de log pincharemos sobre Show rule preferences y definiremos las reglas que nos sean útiles.

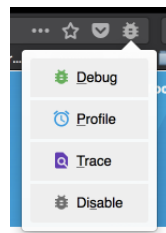


7.3.1.1.3. Firefox

En Mozilla Firefox instalaremos el complemento Xdebug Helper, que nos permitirá lanzar el debug directamente desde el navegador en cualquier punto del proyecto. Para ello buscaremos esta extensión en el addons de Mozilla [<https://addons.mozilla.org/en-US/firefox/addon/xdebug-helper-for-firefox/>] y la añadiremos a nuestro navegador.



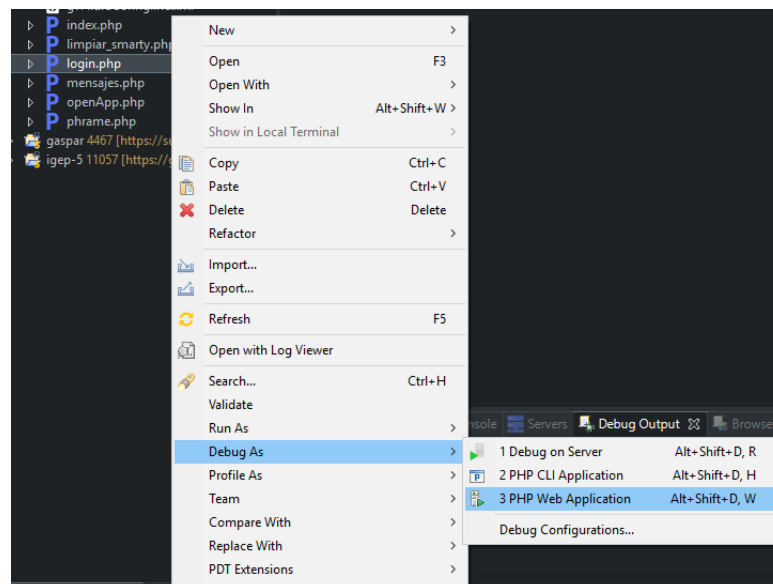
Para usar la extensión, accedemos a nuestro proyecto en el navegador y le damos al botón **Debug**. Navegamos hasta el punto donde hemos colocado un **breakpoint** y eclipse se abrirá automáticamente en modo debug.



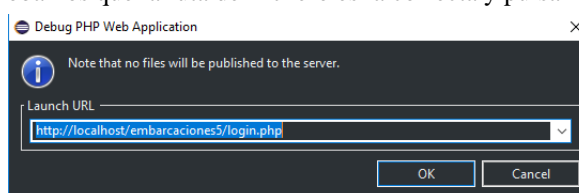
7.3.1.2. Depuración

7.3.1.2.1. Eclipse / PHP

Para iniciar la depuración desde eclipse podemos hacerlo directamente desde el fichero inicial de la aplicación. En nuestro caso este fichero es **login.php**, por tanto hacemos click derecho sobre dicho fichero y seleccionamos **Debug As > PHP Web Application**.

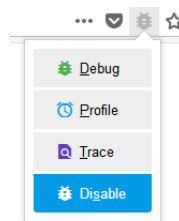


En la siguiente ventana comprobamos que la ruta del fichero es la correcta y pulsamos OK.



Name	XDEBUG_SESSION
Value	XDEBUG_ECLIPSE
Host	localhost
Path	/
Expires	Tue, 12 Nov 2019 08:47:49 GMT
Secure	No
HttpOnly	No

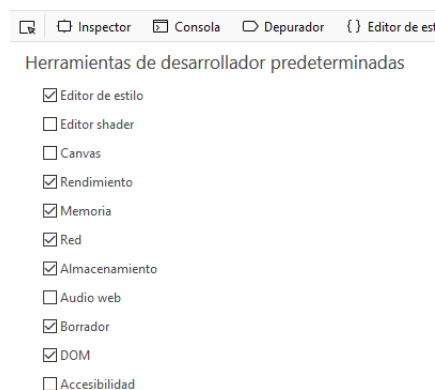
Para detener la depuración lo único que debemos hacer es presionar el botón **Disable**. Esto borrará la cookie de Xdebug del navegador.



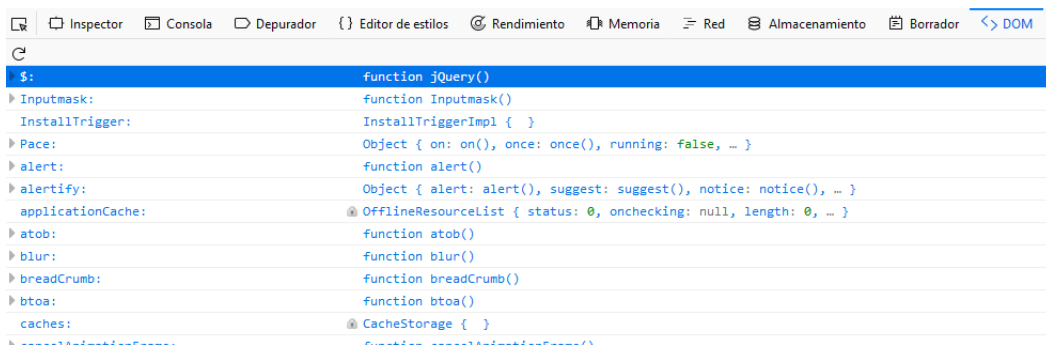
7.3.2. Consola del navegador (Firefox)

7.3.2.1. Firefox / JavaScript

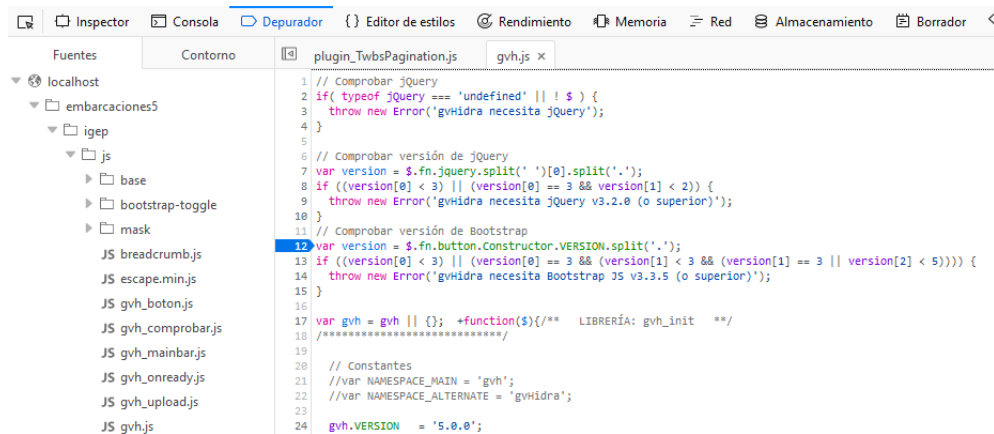
También podemos depurar el javascript que se ejecuta en el navegador. Firefox y casi todos los navegadores disponen de una herramienta de depuración. Pulsando F12 podemos acceder a esta herramienta, una vez en ella pulsamos F1 y se nos abrirá la configuración. En el apartado **Herramientas de desarrollador predeterminadas** marcamos la casilla **Borrador y DOM**.



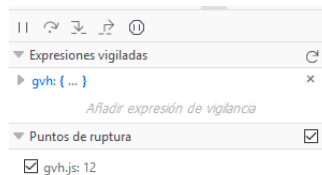
Aparecerán dos pestañas. En Borrador podemos escribir código javascript para depurar la aplicación a nuestro gusto, y en DOM aparecen todas las variables y objetos del entorno, con sus respectivos tipos y valores.



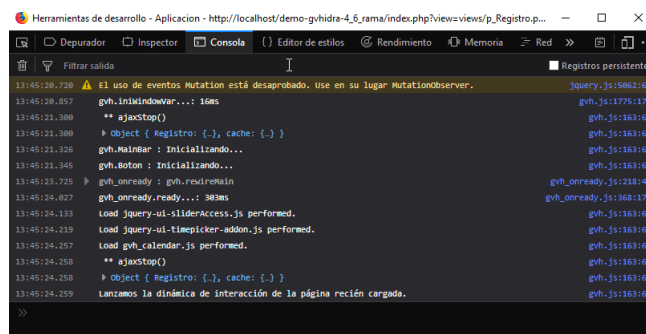
En la pestaña **Depurador** tenemos todos los ficheros JS de la aplicación, y podemos añadir breakpoints como en Eclipse.



Cuando la ejecución llega a dicho punto se para y nos muestra información del entorno, como en la depuración de PHP. En el cuadro **Expresiones vigiladas** podemos filtrar las variables por su nombre, y nos aparecerá su valor en tiempo de ejecución. También disponemos de las opciones de navegación de Eclipse (Step Into, Step over,...).



En la pestaña **Consola** Se verán los mensajes de consola que lleva el framework por defecto, y los que se hayan añadido por necesidad en la aplicación a la hora de desarrollarla.



Hay que entender que el mostrar mensajes en esta consola puede relentizar la ejecución de la aplicación, por lo tanto es interesante no mostrarlos cuando la aplicación esté en producción. La configuración de activar o no el log de JS de la aplicación se hace en el fichero de configuración de la aplicación, externo.gvHidraConfig.inc.xml, con la propiedad **logJSSettings**

Activar:

```
<logJSSettings status='LOG_ALL'>
```

Desactivar:

```
<logJSSettings status='LOG_NONE'>
```

Si se quiere hacer uso de la consola JavaScript en algún fichero JS propio de la aplicación es aconsejable utilizar la función **gvh.showConsoleMsg(type,msg)**. Ya que al hacer uso de este método funcionará el activar o no del log con la variable **logJSSettings**.

Esta función necesita de dos parámetros:

- *type*: Será el tipo de mensaje que se mostrará en la consola. Los valores posibles son los siguientes:
 - 'log': Mensajes generales de información.
 - 'error': Cuando queramos destacar que se ha producido un error, el mensaje en consola aparecerá en rojo.
 - 'warn': Muestra un mensaje de advertencia.
 - 'dir': Muestra un listado interactivo de las propiedades de un objeto JavaScript específico.
 - 'table': Muestra datos tabulares en forma de una tabla.
 - 'beginGroup': Crea un grupo en el que indenta todos los mensajes que se encuentren hasta que se indique cierre del grupo.
 - 'endGroup': Finaliza el grupo abierto con "beginGroup".
- *msg*: Mensaje que queremos que se visualice en la consola.

Ejemplos de uso:

- Error

```
gvh.showConsoleMsg('error',' ¡¡Falta la indicación del campo destino de la ventana de selección!!');
```

- Grupo

```
gvh.showConsoleMsg('beginGroup',' ** gvH_panel.js - changeField() ');
...
gvh.showConsoleMsg('log', ' FIELD: '+idCampo );
...
gvh.showConsoleMsg('error', '¡¡¡ changeField() - '+campoJSON+' no existe este campo en el registro '+regJSON+' !!!' );
...
gvh.showConsoleMsg('endGroup');
```

7.3.3. Debug de plantillas Smarty

Cuando se está ejecutando una aplicación se puede activar la consola de debug de Smarty para la ventana que en ese momento tenemos en pantalla.

Para ello simplemente se tiene que añadir el parámetro "SMARTY_DEBUG" a la ruta URL en la barra del navegador.

Por ejemplo, si la url de la aplicación fuera:

```
http://localhost/app/index.php?view=views/Vista.php
```

Para mostrar la consola de debug habría que añadirle **&SMARTY_DEBUG**.

```
http://localhost/app/index.php?view=views/Vista.php&SMARTY_DEBUG
```

Debido a que se muestran detalles internos de la aplicación, por seguridad este modo de depuración sólo está disponible en los entornos local y de desarrollo. Por lo tanto **NO ESTÁ DISPONIBLE EN ENTORNO DE PRODUCCIÓN**. Esto es, sólo está activado en los casos en que, en el fichero de configuración de la aplicación ('gvHidraConfig.inc.xml') la propiedad 'p_entorno' tenga el valor **LOCALHOST** o **DESARROLLO**:

```
<property id="p_entorno">LOCALHOST</property>
```

```
<property id="p_entorno">DESARROLLO</property>
```

7.4. CUSTOM

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 .

7.4.1. 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>
```

7.4.2. 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...

7.4.3. 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.

7.4.4. 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 librería 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://fontawesome.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;
}
```

7.4.5. 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**

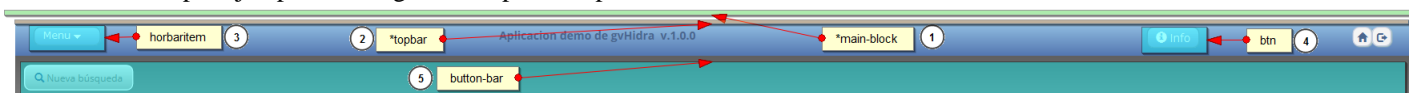
Las terceras son clases propias de Bootstrap pero que se modifican en nuestras hojas de estilos para bien reemplazar estilos que vienen predeterminados o bien para incluir nuevos estilos sobre estas clases, a este tipo de clases las llamaremos clase híbrida.

7.4.6. 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 señalado con una capa de un color, por realizar el proceso de una manera clara en algunos contenedores que ocupen todo el bloque se señalará 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 señalado 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.

7.4.7. 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;
}
```

7.4.8. 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**.

```

/* Propiedades de la capa LOGO */
#gvHFooterLogo_1 {
    background-image: url("../images/logos/logo.gif");
    display: inline-block;
    width: 241px;
    height: 58px;
}

/* Propiedades 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);
}

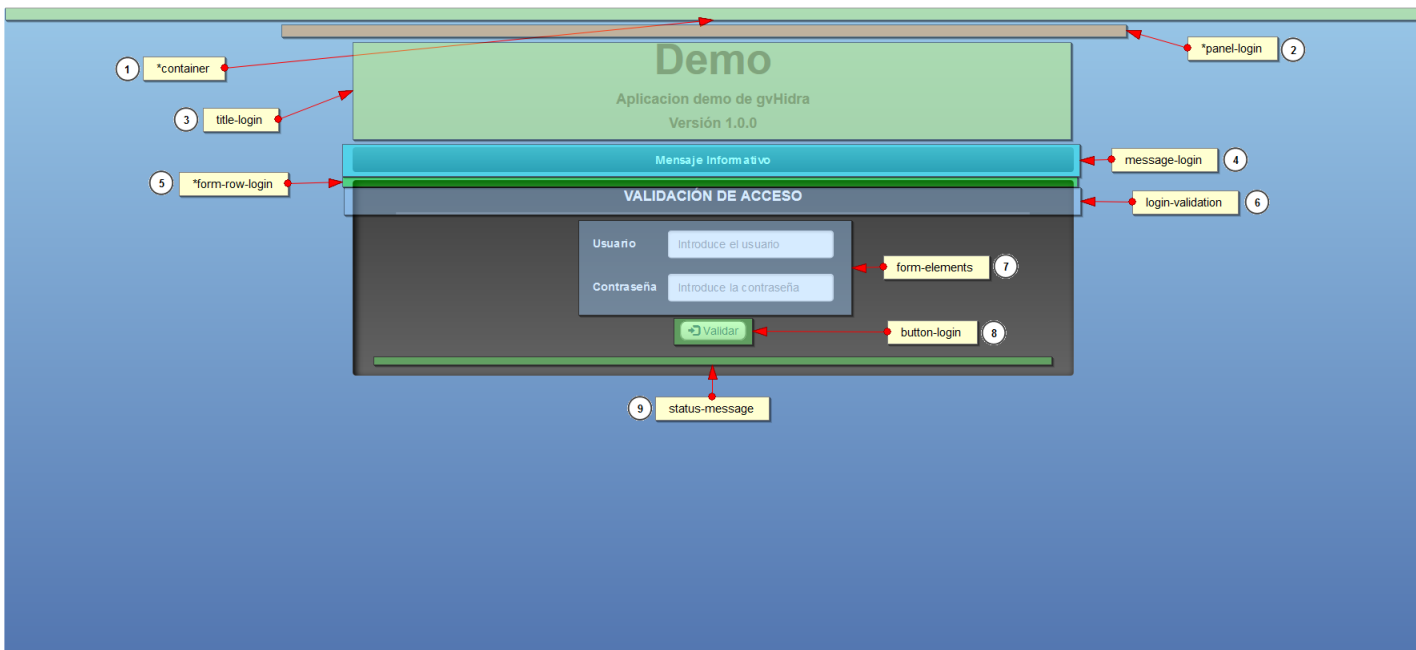
#gvHAboutLogo_2 {
  display: block;
  width: 90px;
  height: 90px;
  background-image: url("../images/miLogodeApp2.png");
  opacity: 1;
  filter: alpha(opacity=100);
}

#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...**

7.4.9. Panel login

Los estilos que usa el panel de login se encuentra en el archivo application-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 pre-determinada, en la hoja de estilos application-login.css modificamos o reemplazamos algunos atributos que vienen de forma pre-determinada.

```
.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

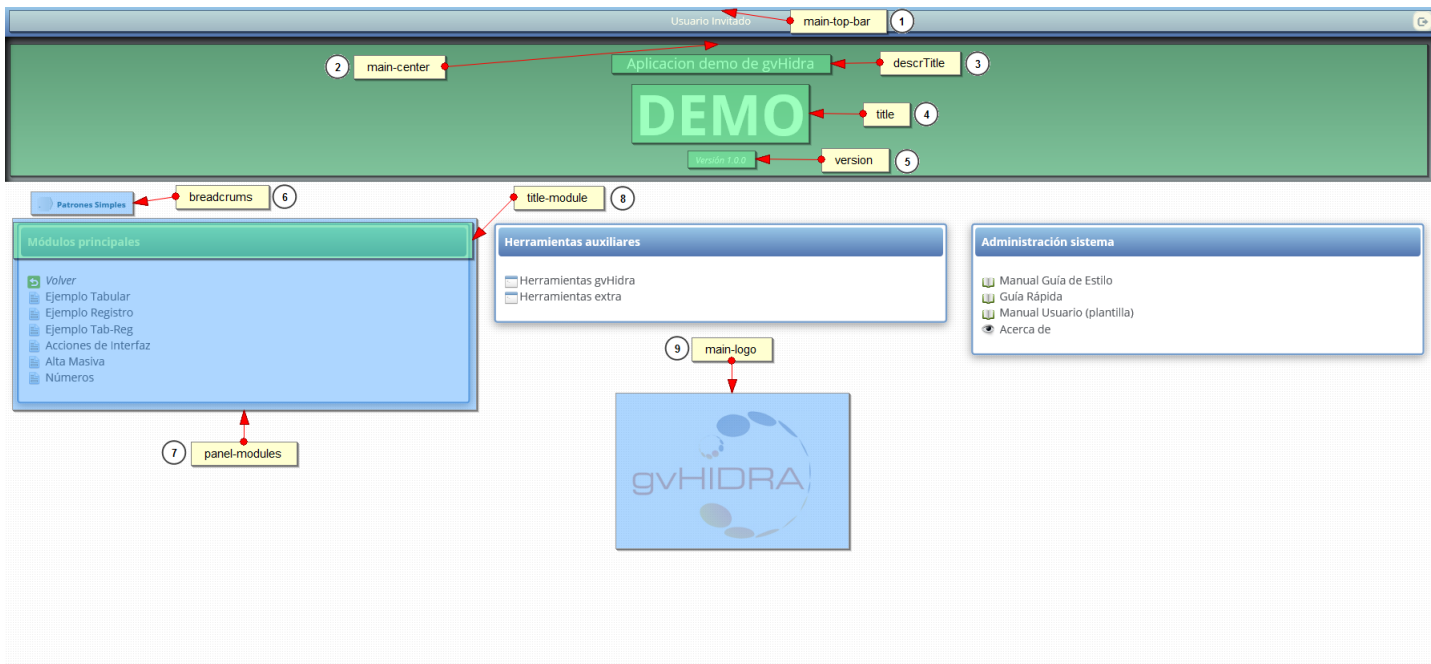
Las clase form-group viene con estilos predeterminados de Bootstrap para los elementos label e input.

Le hemos agregado estilos propios de la aplicación para las etiquetas label del formulario..

```
.status-message {
  color: #ED1B24;
  font-size: 18px;
}
```

7.4.10. 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.

Identificador 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);
}
```


7.4.11. 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**

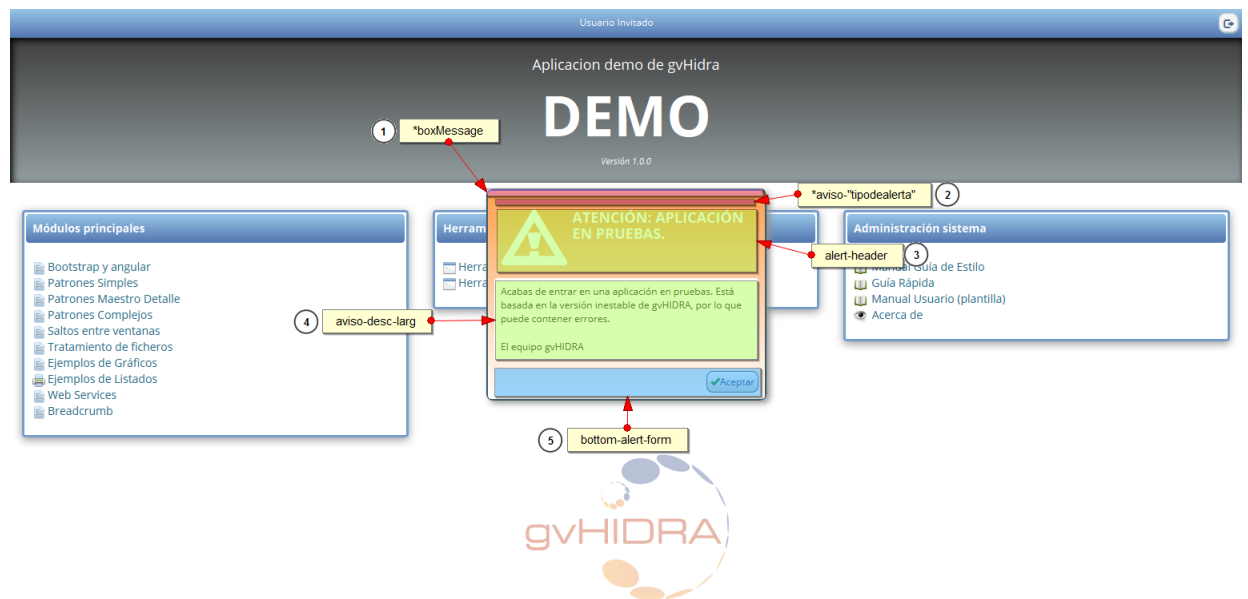
Contenedor que contiene el footer.

Clase híbrida: modal-footer

```
.modal-footer {
  border-top: none;
  color: #fff;
}

.modal-footer img {
  width: 64px;
}
```

7.4.12. 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);
}

.avis-error {
background:linear-gradient(#D0240C,#ffffff);
background:-o-linear-gradient(#D0240C,#ffffff);
background:-moz-linear-gradient(#D0240C,#ffffff);
background:linear-gradient(#D0240C,#ffffff);
}

.avis-sugerencia {
background:linear-gradient(#5F9A2E,#ffffff);
background:-o-linear-gradient(#5F9A2E,#ffffff);
background:-moz-linear-gradient(#5F9A2E,#ffffff);
background:linear-gradient(#5F9A2E,#ffffff);
}

.avis-avis {
background:linear-gradient(#3475D9,#ffffff);
background:-o-linear-gradient(#3475D9,#ffffff);
background:-moz-linear-gradient(#3475D9,#ffffff);
background:linear-gradient(#3475D9,#ffffff);
}

.avis-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

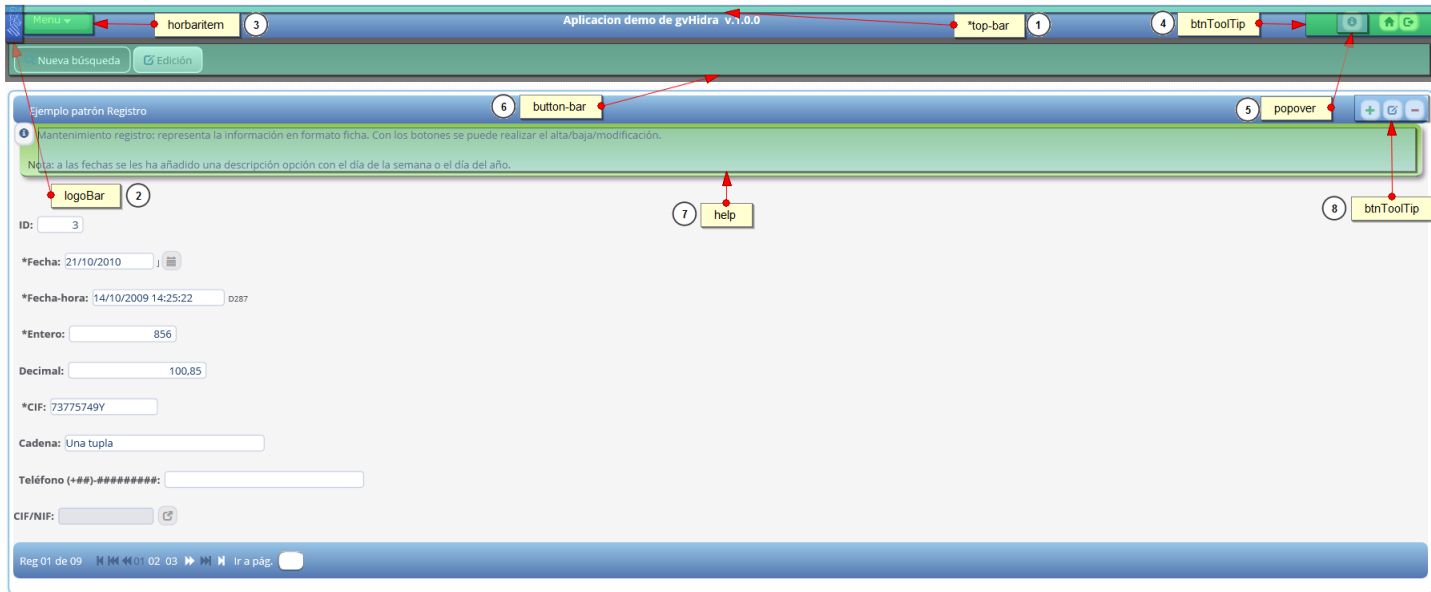
Los estilos para el botón aceptar son los siguientes:

```
.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;
}
```

7.4.13. 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

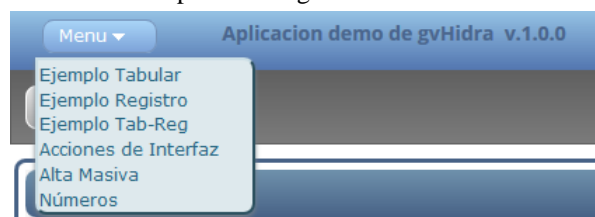
Estilos para el botón:

```
.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;
}

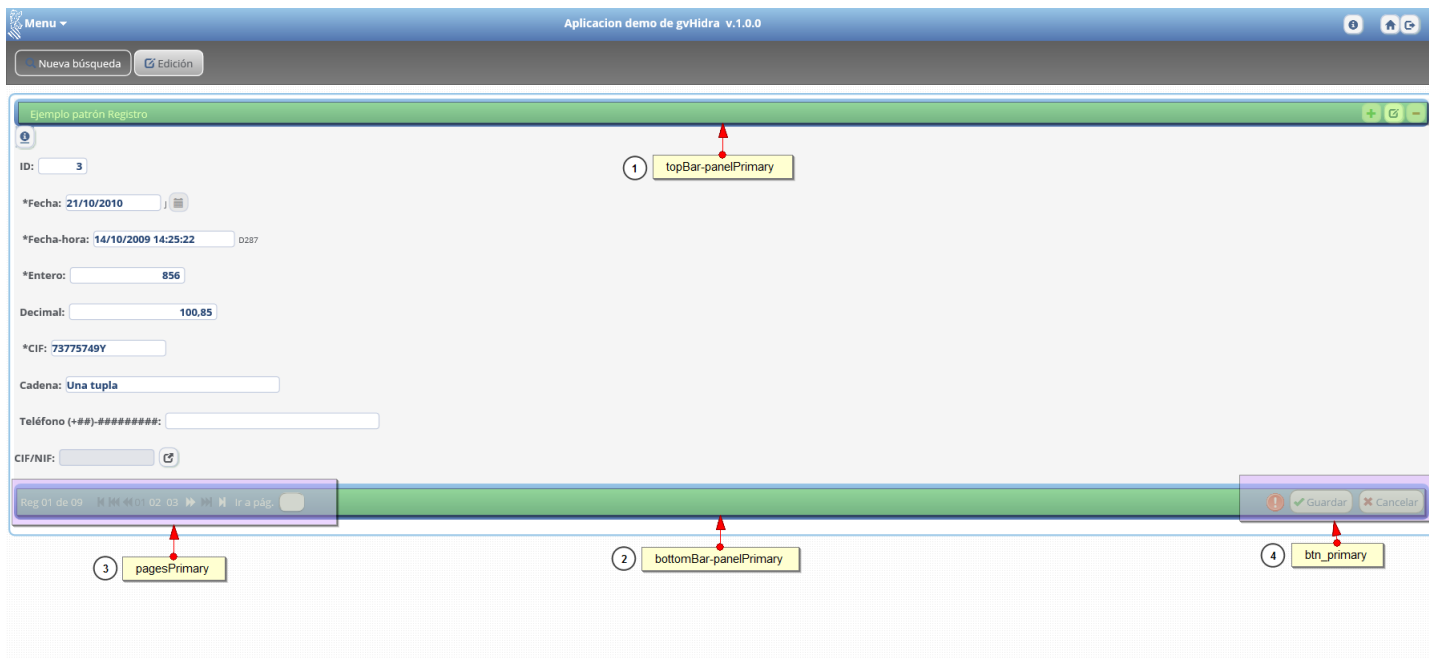
```

```

.btnToolTip.off {
  visibility: hidden
}

```

7.4.14. 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 utilice 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;
}

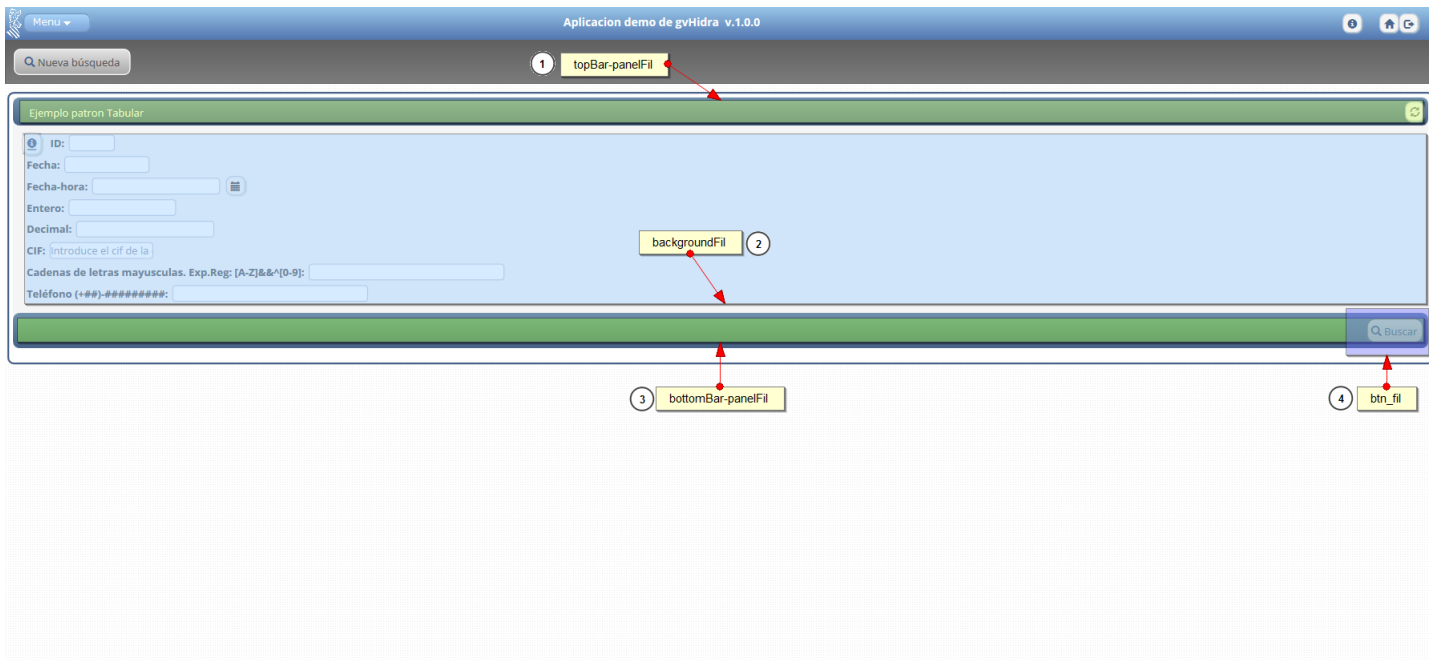
.btn_primary:hover {
  background: linear-gradient(#d8d8d8, #ffffff);
  background: -o-linear-gradient(#d8d8d8, #ffffff);
  background: -moz-linear-gradient(#d8d8d8, #ffffff);
  background: linear-gradient(#d8d8d8, #ffffff);
  border-radius: 10px;
  border: 1px solid;
  border-color: #999999;
}
```

En el caso de que el botón tenga un glyphicon usará los siguiente estilos.

```
.glyphicon-ok:before {
  color: #42a650;
}

.glyphicon-remove:before {
  color: #a74a43;
}
```

7.4.15. 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

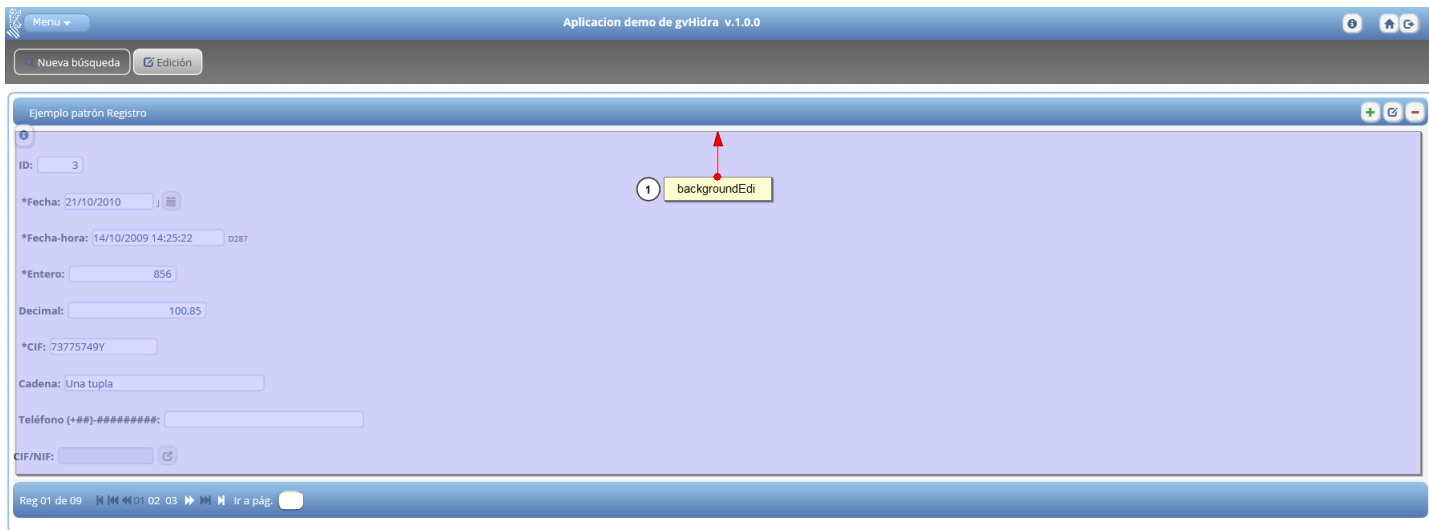
```
.btn_fil {
  color: #4a668e;
  background: linear-gradient(#ffffff, #d8d8d8);
  background: -o-linear-gradient(#ffffff, #d8d8d8);
  background: -moz-linear-gradient(#ffffff, #d8d8d8);
  background: linear-gradient(#ffffff, #d8d8d8);
  border: 1px solid;
  border-color: #777777;
  border-radius: 10px;
  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;
}
```

7.4.16. 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;
}
```

7.4.17. Panel LIS

Aplicación demo de gvHidra v.1.0.0

Ejemplo patron Tabular

ID	Fecha	Fecha-hora	Entero	Decimal	CIF/NIF
3	21/10/2010 J	14/10/2009 14:25:22 D287	856	100.85	
5	06/06/2013 J	08/06/2013 12:33:33 D159	859		
6	08/10/2012 L	15/12/2012 12:31:00 D350	8.989		
7	04/04/2012 x	04/05/2013 00:00:00 D124	46.546		
8	30/06/2012 s	28/07/2012 00:12:00 D210	46.132	1.00	
12	12/12/2012 x	12/12/2012 00:00:00 D847	456		
17	25/08/2012 s	30/08/2012 00:00:00 D243	5.656		
32	04/03/2013 L	04/05/2013 00:00:00 D124	2.132		

(Nº reg. 9) Pg. 01 de 02 Ir a pág.

- 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: oddRecord

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 ↑
<input type="checkbox"/>	01	DICCIONARIO
<input type="checkbox"/>	02	MANUAL
<input type="checkbox"/>	03	TUDIOS2
<input type="checkbox"/>	04	LEGISLACION
<input type="checkbox"/>	05	JURISPRUDENCIA
<input type="checkbox"/>	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;
}
```

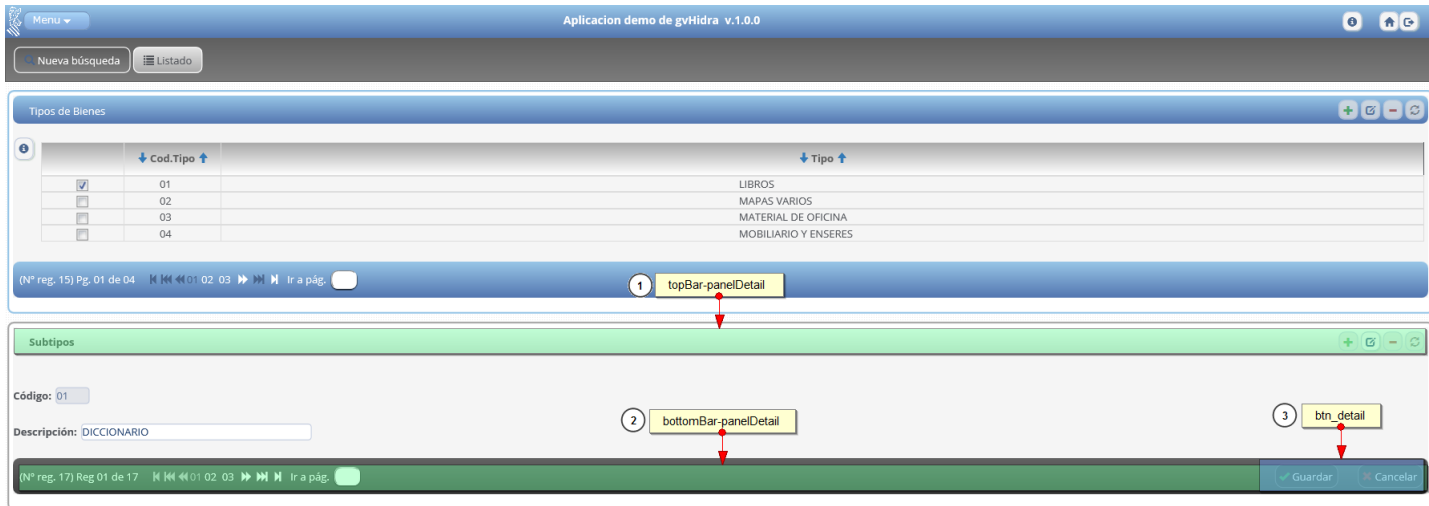
Clase de la aplicación: suggestionRow

Este estilo corresponde al de "sugerencia".

```
.suggestionRow {
  background-color: #ced6b2;
}
```

7.4.18. 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.

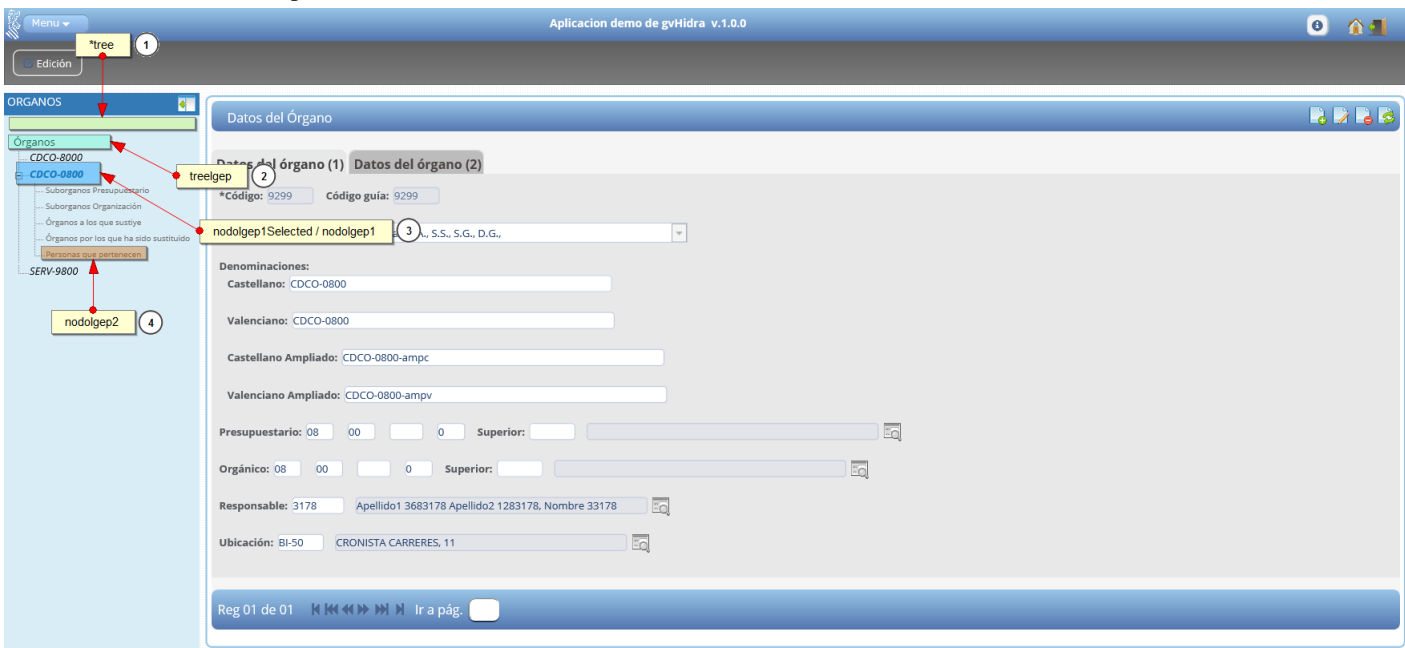
Clase de la aplicación: btn_detail

```
.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);
}
```

7.4.19. Panel árbol

Los estilos de este panel se encuentran 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).

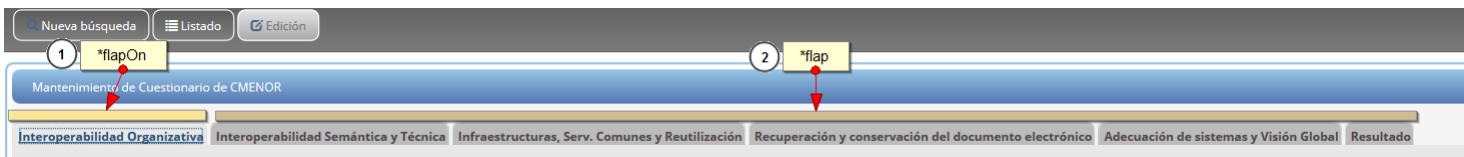
Clase de la aplicación: .nodoIgep1 y nodoIgep1Selected (con sus posibles opciones a:link a:hover ...etc)

• **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)

7.4.20. Solapas



Las solapas tienen 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;
}

```


}

7.4.21. 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);
}
```

7.4.22. 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.

7.4.22.1. Pantalla principal

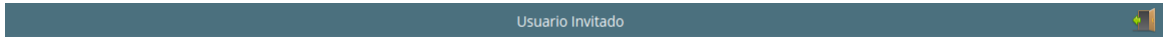
Pantalla actual:



Comenzaremos a realizar los cambios por la parte superior de la pantalla.

Primero se visualizarán 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 #albfbf;
  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;
}
```

7.4.22.2. 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:



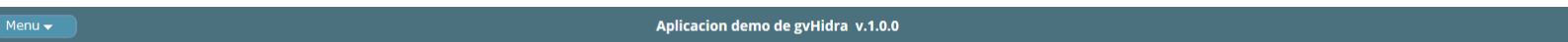
Resultado final con estilos clásicos:



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 layersmenu-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:

emplo 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;  
}
```


Parte V. Apendices

Tabla de contenidos

A. Pluggins, que son y como usarlos en mi aplicación.	221
A.1. Documentación Plugins gvHidra	221
A.1.1. cwarbol	222
A.1.2. cwareatexto	223
A.1.3. cwbarra	224
A.1.4. cwbarrainfpanel	225
A.1.5. cwbarrasuppanel	225
A.1.6. cwboton	226
A.1.7. cwbotontooltip	228
A.1.8. cwcampotexto	232
A.1.9. cwcontenedor	234
A.1.10. cwcheckbox	235
A.1.11. cwficha	236
A.1.12. cwfichaedicion	237
A.1.13. cwfila	237
A.1.14. cwimagen	238
A.1.15. cwinfocontenedor	239
A.1.16. cwinformation	240
A.1.17. cwlabel	241
A.1.18. cwlista	243
A.1.19. cwmarcopanel	245
A.1.20. cwmaps	245
A.1.21. cwmenulayer	247
A.1.22. cwpaginador	248
A.1.23. cwpanel	249
A.1.24. cwpantallaentrada	250
A.1.25. cwrichareatexto	251
A.1.26. cwslider	252
A.1.27. cwsolapa	253
A.1.28. cwtabla	254
A.1.29. cwupload	255
A.1.30. cwventana	256
B. Listados Jasper en gvHIDRA	259
B.1. Integracion de Jasper en un mantenimiento	259
B.2. Depuración de errores al intentar ejecutar informes JasperReports en gvHidra	261
B.3. Recomendaciones y generalidades sobre integración gvHidra y Jasper	263
C. Pasos de migración para pasar a la versión 5.0.0	266
C.1. Codificación del proyecto.	266
C.2. Cambios en la lógica de negocio.	266
C.3. Migración de los ficheros XML de configuración.	267
C.4. Migración del fichero AppMainWindow.php	268
C.5. Migración del fichero mappings.php	268
C.6. Migración de las plantillas (TPL).	268
C.6.1. Plugins obsoletos.	268
C.6.2. Maestro/Detalle.	269
C.6.3. Migración del nombre de los plugins.	269
C.6.4. Notas importantes sobre los parámetros de los plugins.	270
C.6.5. Parámetros de plugins que se deben eliminar.	270
C.6.6. Parámetros de plugins que se deben renombrar.	271
C.6.7. Paso de la variable \$smt_iteracionActual al uso de la librería IgepJS.js	272
C.7. Migración de los views.	275
D. Novedades de la versión	276
D.1. Herencia en plantillas, plugins y ficheros de idiomas.	276
D.2. Mejora de rendimiento en la visualización de resultados.	277

D.3. Nuevos customs.	277
D.4. Menú de la aplicación.	278
D.5. Configuración aspectos generales de las ventanas.	280
D.5.1. Icono "modificado" en barra superior.	280
D.5.2. Ubicación botonera: Nueva búsqueda, listado, edición.	281
D.5.3. Redimensionar paneles.	282
D.5.4. Conteo de registros en las solapas de los detalles.	283
D.6. Configuración nuevas funcionalidades en paneles tabulares.	284
D.6.1. Opciones de selección de registros en un tabular.	284
D.6.2. Selección directamente en la fila.	285
D.6.3. Mostrar número total de registros en un tabular.	286
D.6.4. Alineación de campos en un panel tabular.	287
D.7. Nuevas operaciones y parámetros en plugins.	287
D.7.1. cwinfocontenedor. Operaciones getValue() y setValue().	287
D.7.2. cwinformation. Operaciones getValue() y setValue(). Parámetros nuevos.	288
D.7.3. cwbotontooltip con funcionamiento independiente del estado del panel.	289
D.7.4. Visibilidad de etiquetas que acompañan componentes.	289
D.7.5. Parámetro confirm en botones tooltip.	289
D.8. Debug en desarrollo.	290
D.8.1. Debug de plantillas en tiempo de ejecución en entorno de desarrollo.	290
D.8.2. Debug de JavaScript.	290

Apéndice A. Pluggins, que son y como usarlos en mi aplicación.

A.1. Documentación Plugins gvHidra



Nota

A diferencia de versiones anteriores de gvHidra, la versión 5.x obliga a escribir los **nombres de los plugins en minúsculas** cuando son utilizados en las plantillas (TPL).

Lista de Pluggins

1. cwarbol [222]
2. cwareatexto [223]
3. cwbarra [224]
4. cwbarrainfpanel [225]
5. cwbarrasuppanel [225]
6. cwboton [226]
7. cwbotontooltip [228]
8. cwcampotexto [232]
9. cwcontenedor [234]
10. cwcheckbox [235]
11. cwficha [236]
12. cwfichaedicion [237]
13. cwfila [237]
14. cwgraph [237]
15. cwimagen [238]
16. cwlabel [241]
17. cwinfocontenedor [239]
18. cwinformation [240]
19. cwlista [243]
20. cwmaps [245]
21. cwmarcopanel [245]
22. cwmenulayer [247]
23. cwpaginador [248]
24. cwpanel [249]
25. cwpantallaentrada [250]
26. cwsolapa [253]

27.cwtabla [254]

28.cwupload [255]

29.cwventana [256]

Abreviaturas: **A** = Alfanumérico | **B** = Booleano | **E** = Enumerado | **M** = Matriz | **N** = Numérico

A.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.

- **Pluggins que pueden contener a cwarbol**

- Padres
 - cwmarcopanel

- **Pluggins que puede contener cwarbol**

- Hijos
 - cwpanel

Tabla de argumentos de cwarbol

Nombre	Tipo	Opcional	Descripción	Valores
ancho	N	true	Define el ancho del panel árbol en porcentaje, por defecto es un 25%.	
arbol	A	false	Estructura definida para el árbol (archivo xml). Será una variable smarty (\$smarty_objArbol) que es sustituida por IgepPanel.	
estado	A	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: <ul style="list-style-type: none"> • <i>on</i>: visible y activo • <i>off</i>: visible e inactivo • <i>inactivo</i>: no visible e inactivo
título	A	true	Dará el título al panel que contiene el árbol.	

Ejemplos de uso del plugin cwarbol:

Árbol con dos paneles diferentes. La variable \$smarty_panelVisible es sustituida por IgepPanelArbol, que le asignará el valor del tipo de nodo que se ha seleccionado, coincidirá con el nombre que se le pase como primer parámetro a la función setNodoPanel().

```
{cwarbol estado=$estado_edi arbol=$smarty_objArbol}
{if $smarty_panelVisible == "ANYO" || $smarty_panelVisible == "FACTURA"}
  {cwpanel
    .....
  }
{else}
  {cwpanel
    .....
  }
}
```

```
{/if}
{/cwarbol}
```

Lista de plugins [221]

A.1.2. cwareatexto

Equivalente al TEXTAREA de HTML.

- **Plugins que pueden contener a cwareatexto**
 - Padres
 - cwfila
 - cwficha
 - 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
actualizaA	A	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
class	A	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.
cols	N	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.
editable	A	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.
label	A	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...
nombre	A	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	B	true	Con "true/false" indicaremos si es un campo obligatorio a la hora de completar los datos.
placeholder	A	true	Descripción que aparecerá en el campo si no tiene valor.
showLabel	B	true	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.
tabIndex	N	true	Especifica el orden de tabulación.

Nombre	Tipo	Opcional	Descripción
title	A	true	El valor de este atributo será el que se asignará al equivalente atributo "title" html.
value	A	true	Valor por defecto que aparecera en el campo.
visible	B	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.

Ejemplos de uso del plugin cwareatexto:

Ejemplo: Declaración de cwareatexto.

```
{cwareatexto nombre="comentario" label="Expedientes" tabindex="2" }
```

Lista de plugins [221]

A.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
iconHome	A	true	Icono que aparecerá a la derecha de la barra superior para acceder a la pantalla principal de la aplicación.
iconInfo	A	true	Icono que aparecerá a la derecha de la barra superior que mostrará una ventana de información sobre la aplicación.
iconOut	A	true	Icono que aparecerá a la derecha de la barra superior para salir de la aplicación.
info	array	true	Array con información que define la aplicación (variable smarty \$smarty_info asignada internamente): <ul style="list-style-type: none"> • applicationName: nombre de la aplicación • customTitle: título de la aplicación • barTitle: si este campo tiene valor, sobrescribirá el título del parámetro "customTitle" • version: versión de la aplicación • usuario: usuario validado en la aplicación • entorno: entorno de ejecución de la aplicación

Ejemplos de uso del plugin cwbarra:

Ejemplo: Una barra con Menú (plugin **cwmenulayer**).

```
{cwbarra info=$smtty_info iconOut="glyphicon glyphicon-log-out" iconHome="glyphicon glyphicon-home" iconInfo="glyphicon glyphicon-info-sign"}
  {cwmenulayer name="$smtty_nombre" arrayMenu="$smtty_arrayMenu"}
{/cwbarra}
```

Lista de plugins [221]

A.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 iconCSS="glyphicon glyphicon-search" label=#gvhlang_buscar# class="button"
  accion="buscar" mostrarEspera="true"}
{/cwbarrainfpanel}
```

Lista de plugins [221]

A.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
flotante	B	true	Cuando se active la barra superior del panel será flotante. Esto será útil en el caso de que un panel sea más largo que lo visible en pantalla, al utilizar la barra de des-

Nombre	Tipo	Opcional	Descripción
			plazamiento la barra superior del panel se irá desplazando quedando siempre en la parte superior de la pantalla. Este parámetro no es válido para ventanas modales.
layout	A	true	Puede contener los siguientes valores. <ul style="list-style-type: none"> <i>no-maximize</i>: Oculta el botón de maximizar/restaurar el tamaño del panel. <i>show-modified</i>: Muestra el icono de registro modificado (en aquellos casos en los que el usuario modifique el registro), con comportamiento análogo al icono de modificación que ya se mostraba en "cwbarrainfpanel".
title	A	true	Establece la descripción de la acción del panel.

Ejemplos de uso del plugin cwbarrasuppanel:

Ejemplo: Declaración de cwbarrasuppanel.

```
{cwbarrasuppanel titulo="BUSCAR ESTADOS"}
{cwbotontooltip iconCSS="glyphicon glyphicon-refresh" title=#gvhlang_limpiar#
accion="limpiar" actuaSobre="ficha"}
{/cwbarrasuppanel}
```

Lista de plugins [221]

A.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
 - cwbarrasuppanel

• **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
accion	A	true	Especifica la acción que realiza el botón	Puede tomar los siguientes valores: <ul style="list-style-type: none"> • <i>guardar</i>: Ejecuta la acción del panel. • <i>cancelar</i>: Cancela la acción del panel. • <i>saltar</i>: La acción es un salto. • <i>volver</i>: Se retorna de un salto. • <i>listar</i>: La accion invoca a un listado.

Nombre	Tipo	Opcional	Descripción	Valores
				<ul style="list-style-type: none"> <i>cancelarvs</i>: Para un botón Cancelar de una ventana emergente. <i>aceptarvs</i>: Sólo para el botón Aceptar de la ventana de selección. <i>particular</i>: El botón realiza una función que no es genérica. En este caso el parámetro "action" corresponderá a la acción definida en el método "accionesParticulares" de la clase manejadora correspondiente.
action	A	true	Acción que queremos que se ejecute al pulsar el botón y que sea diferente a la de por defecto del panel.	
action-Cancel	A	true	Este parametro indicará la acción que se debe ejecutar en un botón cancelar en un mensaje de confirmación.	
actuaSobre	A	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.	
checkForm	B	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']	
claseManejadora	A	true	Será necesario si el botón no se encuentra bajo ningún panel correspondiente a una clase manejadora, por ejemplo en la barra superior.	
claseManejadoraOrigen	A	true	Parámetro OBLIGATORIO SÓLO cuando es un botón de la VENTANA DE SELECCIÓN . Indica la clase manejadora del panel origen.	
class	A	true	Contendrá el css personalizado que se le quiera dar al botón.	
confirm	B	true	Con este parámetro aparecerá una ventana emergente de confirmación para ejecutar la acción del panel.	
editable	E	true	Puede interesarnos inhabilitar un botón para determinados perfiles de usuario o datos.	
filaActual	N	true	Nos indica la fila seleccionada, SÓLO se utiliza en la VENTANA DE SELECCIÓN donde es OBLIGATORIO.	
formActua	A	true	Parámetro OBLIGATORIO SÓLO cuando es un botón de la VENTANA DE SELECCIÓN . Indica el nombre del formulario en el que se de-	

Nombre	Tipo	Opcional	Descripción	Valores
			volverán los valores elegidos en la ventana de selección.	
funcion	A	true	Este parametro se utiliza por si queremos añadirle funcionalidad extra al botón. En caso de utilizarse no debe añadirse la palabra reservada "javascript:".	
id	A	true	Identificador para el botón, sólo en el caso de que la acción sea "particular".	
iconCSS	A	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	
label	A	true	Fija el texto descriptivo que aparece en el botón.	
modoPanelOrigen	A	true	Parámetro OBLIGATORIO SÓLO cuando es un botón de la VENTANA DE SELECCIÓN . Modo en el que se encuentra el panel origen Modificación, inserción, lectura.	
open-Window	B	true	Indica si queremos que la acción del botón la ejecute en una ventana diferente.	
panelActua	A	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.	
panelOn	A	true	Será necesario si el botón no se encuentra bajo ningún panel, por ejemplo en la barra superior.	
tabindex	N	true	Orden de tabulación..	
title	A	true	Este parametro se utiliza para visualizar la ayuda en linea al situar el puntero del ratón sobre el componente.	
visible	B	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.	

Ejemplos de uso del plugin cwboton:

Ejemplo: Declaración de cwboton.

```
{cwboton iconCSS="glyphicon glyphicon-ok" label="Guardar" class="button"
accion="guardar" }
```

Lista de plugins [221]

A.1.7. cwbotontooltip

Estos componentes se utilizan 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.

- **Pluggins que pueden contener a cwbotontooltip**

- Padres
 - cwbarrasuppanel
 - cwficha
 - cwfila
 - cwsolapa
- **Pluggins que puede contener cwbotontooltip**
 - Hijos
 - El plugin cwbotontooltip es una hoja (no contiene otros pluggins)

Tabla de argumentos de cwbotontooltip

Nombre	Tipo	Opcional	Descripción	Valores
accion	E	true	Indica la función que va a tener asociada ese BotonTooltip (es decir, el tipo de botón que es)	Acción que realizará el botón: <ul style="list-style-type: none"> • <i>insertar</i>: Un botonTooltip de inserción o nuevo • <i>modificar</i>: BotonTooltip de edición o modificación • <i>eliminar</i>: BotonToltip de marcado para borrar • <i>limpiar</i>: BotonToltip de restablecimiento de valores • <i>abrirVS</i>: Botón que abre una Ventana de selección • <i>buscarVS</i>: Botón que ejecuta la búsqueda dentro de la ventana de selección • <i>actualizaCampos</i>: Botón que actualizará una serie de campos • <i>print</i>: imprimirá lo visible en la ventana actual. • <i>exportCSV</i>: exporta a CSV el contenido del panel • <i>ayuda</i>: Botón que nos abrirá el manual de ayuda en una ventana emergente. • <i>openDoc</i>: 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 debe ser el nom-

Nombre	Tipo	Opcional	Descripción	Valores
				bre del campo correspondiente.
action	A	true	Se utiliza para redireccionar en el mapping, la lógica de negocio. IMPORTANTE: Botón insertar en panel de búsqueda action siempre valdrá "nuevo".	
action-Cancel	A	true	Este parametro indicará la acción que se debe ejecutar en un botón cancelar en un mensaje de confirmación.	
actuaSobre	A	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: <ul style="list-style-type: none"> • <i>tabla</i>: si los datos se muestran sobre una tabla y las operaciones se realizan sobre ella • <i>ficha</i>: si los datos se muestran sobre una ficha y las operaciones se realizan sobre ella
claseManejadora	A	true	Será necesario si el botón no se encuentra bajo ningún panel correspondiente a una clase manejadora, por ejemplo la barra superior.	
columnAlign	A	true	En el caso de encontrarse en una tabla, indicará la alineación del componente en la columna. Valores: ['column-align-left', 'column-align-right', 'column-align-center']	
confirm	B	true	Con este parámetro aparecerá una ventana emergente de confirmación para ejecutar la acción del panel.	
checkForm	B	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']	
dependPanel	B	true	Este parámetro permite decidir si el botón es dependiente del estado del panel o no. Si tiene el valor "false", el botón estará activo siempre, es decir no dependerá de que el panel esté en modo inserción o edición.	
editable	E	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.	
external	B	true	Indicará si el botón se encuentra en un panel tabular pero en la zona externa de la tabla.	

Nombre	Tipo	Opcional	Descripción	Valores
filaActual	N	true	Nos indica la fila seleccionada, SÓLO se utiliza en la VENTANA DE SELECCIÓN donde es OBLIGATORIO .	
for	alfaA	true	Este parámetro se utilizará en el caso de que el botón acompañe a un campo, el valor de este parámetro corresponderá con el id del campo.	
formActua	A	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.	
iconCSS	A	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	
id	A	false	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 <i>openDoc</i> el id debe ser el nombre del campo de la consulta que contendrá la ruta al documento.
label	A	true	Etiqueta asociada al botón, que aparecerá al lado de la imagen css como ocurre en el cwboton.	
modoPanelOrigen	A	true	Parámetro OBLIGATORIO SÓLO cuando es un botón de la VENTANA DE SELECCIÓN . Modo en el que se encuentra el panel origen Modificación, inserción, lectura.	
openWindow	B	true	Indica si queremos que la acción del botón la ejecute en una ventana diferente.	
panelActua	A	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.	
panelOn	A	true	Será necesario si el botón no se encuentra bajo ningún panel, por ejemplo en la barra superior.	
rutaManual	A	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.	
showLabel	B	true	Indicará si se debe o no mostrar la etiqueta asociada al botón. Solamente en el caso de que el botón se encuentre dentro de un panel.	
tabindex	N	true	Orden de tabulación.	
title	A	true	Este parametro se utiliza para visualizar la ayuda en linea al situar el puntero del ratón sobre el componente.	
visible	B	true	Con "true/false" indicaremos que queremos forzar si queremos que el botón sea visible/invisi-	

Nombre	Tipo	Opcional	Descripción	Valores
			ble desde el principio. En lugar de obedecer el comportamiento prefijado por gvHidra.	

Ejemplos de uso del plugin cwbotontooltip:

Ejemplo: Botón de eliminar que se encuentra en un panel tabular e insertará en él mismo.

```
{cwbotontooltip iconCSS="glyphicon glyphicon-minus" accion="eliminar"
actuaSobre="tabla" label="Eliminar" }
```

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 iconCSS="glyphicon glyphicon-plus" accion="insertar"
actuaSobre="ficha" action="nuevo" label="Insertar" }
```

```
$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 id="gp" iconCSS="glyphicon glyphicon-plus" accion="abrirVS"
actuaSobre="edi_PARLAMENTARIOS" label="Añadir Grupo" }
```

Lista de plugins [221]

A.1.8. cwcampotexto

Equivalente al INPUT de HTML.

- **Plugins que pueden contener a cwcampotexto**
 - Padres
 - cwcontenedor
 - cwfila
 - cwficha
 - 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
actualizaA	A	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
auto-complete	N	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.

Nombre	Tipo	Opcional	Descripción
autofocus	B	true	Al indicar este parámetro a "true" el campo correspondiente será el que tenga el foco al acceder en el panel.
class	A	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.
columnAlign	A	true	En el caso de encontrarse en una tabla, indicará la alineación del componente en la columna. Valores: ['column-align-left', 'column-align-right', 'column-align-center']
conUrl	B	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.
dataType	M	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
dependPanel	B	true	Este parámetro solamente tiene sentido cuando el campo es de tipo hora o tipo fecha, ya que afectará al botón tooltip que aparece al lado del campo para visualizar el calendario. Si el parámetro tiene el valor "false", el botón estará activo de forma independiente del estado del panel (edición, inserción).
editable	E	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 cwfecha. Si no se especifica el atributo, el campo es editable.
label	A	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 campos obligatorios, etc...
nombre	A	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	B	true	Indicará si el campo es obligatorio que tenga valor a la hora de edición o inserción.
oculto	B	true	Añadido a partir de la versión 1.20. Se utiliza para crear cwcampotextos tipo hidden, pero que pueden ser útiles para campos calculados, etc...
placeholder	A	true	Descripción que aparecerá en el campo si no tiene valor.
showLabel	B	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 es utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
size	A	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.
tabIndex	N	true	Especifica el orden de tabulación.
value	A	true	Valor que queremos que tenga el campo cuando estamos en una inserción.
visible	B	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.

Ejemplos de uso del plugin cwcampotexto:

Ejemplo de uso del atributo máscara:


```
{cwcampotexto nombre="ediCif" size="9" editable="true" label="CIF" dataType=$smtty_dataType_Personas.ediCif}
```

Lista de plugins [221]

A.1.9. 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" label="Fecha
recepcion inicial:"}
{cwcampotexto nombre="filFechaFin" size="10" editable="true" label="Fecha
recepcion final:"}
{/cwficha}
{/cwcontenedor}
```

Lista de plugins [221]

A.1.10. cwcheckbox

Equivalente al CHECKBOX de HTML.

- **Plugins que pueden contener a cwcheckbox**

- Padres
 - cwcontenedor
 - cwfila
 - cwficha
 - cwsolapa

- **Plugins que puede contener cwcheckbox**

- Hijos
 - El plugin cwcheckbox es una hoja (no contiene otros plugings)

Tabla de argumentos de cwcheckbox

Nombre	Tipo	Opcional	Descripción
actualizaA	A	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
class	A	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.
columnAlign	A	true	En el caso de encontrarse en una tabla, indicará la alineación del componente en la columna. Valores: ['column-align-left', 'column-align-right', 'column-align-center']
dataType	M	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
editable	E	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 cwcheckbox sea hijo de cwfila o cwficha. Si no se especifica el atributo, el campo es editable.
label	A	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...
nombre	A	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	B	true	Indicará si el campo es obligatorio que tenga valor a la hora de edición o inserción.
showLabel	B	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.
size	A	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.
tabIndex	N	true	Especifica el orden de tabulación.

Nombre	Tipo	Opcional	Descripción
title	A	true	Este parametro se utiliza para visualizar la ayuda en linea al situar el puntero del ratón sobre el componente.
value	A	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.
visible	B	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.

Ejemplos de uso del plugin cwcheckbox:

Declaración de cwcheckbox como campo de una tabla.

```
{cwcheckbox label="estado" nombre="descEstado" editable="true" dataType=$dataType_claseManejadora.descEstado}
```

SUBIR [221]

A.1.11. 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
 - 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 datos=$smtty_datosTablaM}
{cwficha}
{cwcampotexto nombre="anyo" editable="true" size="4" value=$smtty_anyoNuevo
label="A&ntilde;o"}
{cwcampotexto nombre="nfactura" editable="false" size="6" value="0"
label="N&uacute;mero Factura"}
```

```
<br>
{cwlista nombre="procedencia" editable="true" value=
$smarty_datosPreInsertadosTinvEntradas2.procedencia label="Procedencia" }
{/cwficha}
{cwpaginador enlacesVisibles="3" }
{/cwfichaedicion}
```

Lista de plugins [221]

A.1.12. 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
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 (\$smarty_datos) que la sustituye IgepPanel.

Ejemplos de uso del plugin cwfichaedicion:

```
{cwfichaedicion datos=$smarty_datosFicha}
{cwficha}
...
{/cwficha}
{/cwfichaedicion}
```

Lista de plugins [221]

A.1.13. 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
onRowClick	A	true	Parámetro para poder seleccionar la fila haciendo click en cualquier parte de la fila. Tendrá el valor del id del componente que se quiera lanzar la acción.
on-RowDbClick	A	true	Parámetro para poder seleccionar la fila haciendo doble click en cualquier parte de la fila.
tipoListado	B	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" label="CIF" dataType=
  $dataType_Registro.lisCif}
  {cwcampotexto nombre="lisOrden" size="2" label="Orden" dataType=
  $dataType_Registro.lisOrden}
  {cwlista nombre="lisDepartamento" label="Departamento" value=
  $defaultData_Registro.lisDepartamento dataType=$dataType_Registro.lisOrden}
{/cwfila}
```

Lista de plugins [221]

A.1.14. 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
alt	A	true	Texto alternativo que se representa cuando la imagen no puede ser mostrada.
bumpbox	B	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.
height	N	true	Indica el alto de la imagen.
label	A	false	Texto que acompañará a la imagen.
nombre	A	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 defini-

Nombre	Tipo	Opcional	Descripción
			do por el programador en el atributo matching de la clase correspondiente en la lógica de negocio.
rutaAbs	B	true	Ruta absoluta. Indica que debe buscar la imagen en la ruta absoluta del servidor.
showLabel	B	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 es utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
src	N	true	Ruta a la imagen. Cuando se ha de mostrar una imagen de la bd no es obligatorio poner este parámetro.
title	A	true	Este parametro se utiliza para visualizar la ayuda en linea al situar el puntero del ratón sobre el componente.
visible	B	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.
width	N	true	Indica el ancho de la imagen.

Ejemplos de uso del plugin cwimagen:

```
{cwimagen nombre="fichero" src="imagenes/imagen0.gif" rutaAbs="true"}
```

Lista de plugins [221]

A.1.15. 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**

Nombre	Tipo	Opcional	Descripción
class	A	false	Si queremos particularizar el estilo del componente, aquí se indicará la regla CSS asociada que se deberá haber definido en la hoja de estilos de la aplicación.
id	A	false	Nombre para identificar la instancia del componente.
value	A	true	Corresponderá con el contenido que se mostrará en el bloque informativo. Este parámetro es opcional, será necesario cuando se quiera hacer uso de los métodos getValue() y setValue() desde la clase manejadora. Admite sintaxis HTML.

Ejemplo de uso del plugin cwinfocontenedor sin parámetros:

```
{cwinfocontenedor}
... texto ...
{/cwinfocontenedor}
```

Ejemplo de uso del plugin cwinfocontenedor dinámico:

```
{CWInfoContenedor id="info" value="Listado de personas
<b>con acceso</b>
."}{/CWInfoContenedor}
```

Para obtener el contenido del cwinfocontenedor (value) desde la clase manejadora se pueden utilizar los métodos `getValue()` y `setValue()` añadiendo el segundo parámetro, 'external'.

```
$objDatos->getValue( "info", "external" );
$objDatos->setValue( "info", "Listado de personas
<b>sin acceso</b>
" );
```

Lista de plugins [221]

A.1.16. 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 indicado. 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
class	A	true	Con este parámetro se le podrá asociar un estilo propio.
editable	B	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 cwcheckbox sea hijo de cwfila o cwficha. Si no se especifica el atributo, el campo es editable.
iconCSS	A	false	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
label	A	true	Texto que acompañará al botón tooltip.
nombre	A	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.
posicion	E	true	Opciones: ["top", "right", "bottom", "left"]. Posición respecto al botón donde se quiera que aparezca el layer con la información.
size	N	true	Valor para poder asignar tamaño a la columna en el caso de estar en un panel tabular.
trigger	E	true	Opciones: ["focus", "hover"]. Momento en el que se mostrará el texto informativo.
value	A	false	Texto que queremos que aparezca en el bocadillo cuando se haga click en la imagen.
visible	B	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.

Ejemplos de uso del plugin cwinformation:

```
{cwinformation iconCSS="glyphicon glyphicon-plus" nombre="cif" value="00000" }
```

Para obtener el contenido del cwinformation (value) desde la clase manejadora se pueden utilizar los métodos **getValue()** y **setValue()**.

Lista de plugins [221]

A.1.17. 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	Valores
class	A	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.	
columnAlign	A	true	En el caso de encontrarse en una tabla, indicará la alineación del componente en la columna. Valores: ['column-align-left', 'column-align-right', 'column-align-center']	
editable	E	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.	
iconCSS	A	false	Contendrá la etiqueta CSS correspondiente al icono que queremos que se muestre.	
label	A	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.	
newWindow	B	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.	
nombre	A	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	

Nombre	Tipo	Opcional	Descripción	Valores
			el atributo matching de la clase correspondiente en la lógica de negocio.	
positionIcon	E	true	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, por defecto el icono aparecerá a la izquierda de la etiqueta.	Valores posibles: <ul style="list-style-type: none"> • left • right • up • down
showLabel	B	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.	
size	N	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.	
tipo	E	false	Indicará el tipo de etiqueta que se quiere mostrar.	Valores posibles: <ul style="list-style-type: none"> • icon: icono • iconLabel: icono + etiqueta • label: etiqueta • link: enlace • iconLink: icono + enlace
txtLink	A	true	En el caso de que se opte por un enlace y queramos que la etiqueta sea un texto y no la url correspondiente.	
value	A	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.	
visible	B	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.	

Ejemplos de uso del plugin cwlabel:

(1) Etiqueta <label> de HTML

```
{cwlabel nombre="ediLabel" tipo="label" 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" tipo="iconLabel" iconCSS="glyphicon glyphicon-eye" positionIcon="right" class="classLabel" size="10" editable="true" value=$defaultData_MiClase.ediLabel dataType=$dataType_MiClase.ediLabel}
```

(3) Etiqueta icono css ()

```
{cwlabel nombre="ediLabel" tipo="icon" iconCSS="glyphicon glyphicon-eye" size="10" editable="true" visible="true" value=$defaultData_MiClase.ediLabel dataType=$dataType_MiClase.ediLabel}
```

(4) Etiqueta url

```
{cwlabel nombre="ediLabel" tipo="link" url="http://www.google.es" size="10" editable="true" visible="true" value=$defaultData_MiClase.ediLabel dataType=$dataType_MiClase.ediLabel}
```

Lista de plugins [221]

A.1.18. 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
actualizaA	A	true	Se utiliza este campo en el caso de listas dependientes. Se pondra el nombre del campo que actualizas cuando este toma un valor.
auto-complete	B	true	Al activar este atributo (true), la lista pasara a tener un comportamiento "autocomplete", es decir, tendrá la funcionalidad de que a medida que el usuario escribe irán apareciendo aquellos términos que contengan el texto escrito. En este caso, por detrás se está utilizando la funcionalidad del plugin select2 (https://select2.org/), en el proyecto gvHIDRA se incluyen las librerías.
class	A	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.
columnAlign	E	true	En el caso de encontrarse en una tabla, indicará la alineación del componente en la columna. Valores: ['column-align-left', 'column-align-right', 'column-align-center']
datos	M	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

Nombre	Tipo	Opcional	Descripción
dataType	M	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
editable	E	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.
label	A	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.
multiple	B	true	Si se especifica la lista de selección se convierte en una lista de selección múltiple.
nombre	A	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.
numCaracteres	N	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.
obligatorio	B	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.
showLabel	B	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 es utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
size	N	true	Numero máximo de opciones del desplegable sin que aparezca el scroll.
tabIndex	N	true	Especifica el orden de tabulación.
value	M	false	Matriz con una estructura 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 (\$smarty_datosPreinsertados[claseManejadora]) que será sustituida por IgepPanel. Si el componente se encuentra 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.
visible	B	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.

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 label="Provincia" }
{cwlista nombre="ediCodMun" size="3" editable="true" dataType=
  $dataType_Registro.ediCodMun value=$defaultData_Registro.ediCodMun
  labe="Municipio" }
```

Lista de plugins [221]

A.1.19. cwmarcopanel

Se utiliza para agrupar paneles, cada panel representa un modo de trabajo (búsqueda, ficha, browse...) sobre un mismo conjunto de datos o un subconjunto de los mismos. Sus hijos serán siempre uno o más paneles.

- **Plugins que pueden contener a cwmarcopanel**

- Padres
 - cwventana

- **Plugins que puede contener cwmarcopanel**

- Hijos
 - cwpanel

Este plugin no tiene parámetros.

Tabla de argumentos de cwmaps

Nombre	Tipo	Opcional	Descripción
conPes-tanyas	B	true	Con este parámetro se indicará si se quiere visible o no la botonera ['nueva búsqueda','listado','edición']. Por defecto tendrá el valor "true".

Ejemplos de uso del plugin cwmarcopanel:

Ejemplo: Uso del componente cwmarcopanel en un Maestro-Detalle.

```
{cwventana ...}
...
{cwmarcopanel}
<!--***** PANEL fil *****-->
{cwpanel id="fil" action="buscar" estado="$estado_fil"
  claseManejadora="Registro"}
...
{/cwpanel}
<!--***** PANEL lis *****-->
{cwpanel id="edi" action="operarBD" estado="$estado_edi"
  claseManejadora="Registro"}
...
{/cwpanel}
{/cwmarcopanel}
{/cwventana}
```

Lista de plugins [221]

A.1.20. cwmaps

El plugin cwmaps dibuja un botón en el panel que esté incluido, que a pulsándolo mostrará el mapa con el punto seleccionado y/o polígono, según se decida en la programación. Ese botón necesita de unos parámetros (coordenadas y action) para realizar la llamada al mapa.

- **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
action	A	false	Indica la acción particular definida en la clase manejadora que se debe ejecutar.
coordX	A	false	Indica el nombre del campo de la tpl que contendrá la coordenada X (longitud).
coordY	A	false	Indica el nombre del campo de la tpl que contendrá la coordenada Y (latitud).
label	A	false	Texto que acompañará a la imagen.
nombre	A	false	Nombre para identificar la instancia del componente.
zoom	N	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.

Uso del plugin cwmaps:

Incluir en la tpl correspondiente el plugin cwmaps.

```
{cwcampotexto label="Coord. X" nombre="edi_AR_UTME" size="18"
  editable="true" value=$defaultData_ClaseManejadora.edi_AR_UTME dataType=
  $dataType_ClaseManejadora.edi_AR_UTME}
{cwcampotexto label="Coord. Y" nombre="edi_AR_UTMN" size="18"
  editable="true" value=$defaultData_ClaseManejadora.edi_AR_UTMN dataType=
  $dataType_ClaseManejadora.edi_AR_UTMN}
{cwmaps nombre="visor" coordX="edi_AR_UTME" coordY="edi_AR_UTMN" zoom="12"
  action="ClaseManejadora_poligono" label="MAPA"}
```

En la clase manejadora hay que definir la acción particular indicada en el plugin, "poligono":

```
public function accionesParticulares($str_accion, $objDatos) {
  switch($str_accion) {
    case 'poligono':
      $tupla = IgepSession::dameTuplaSeleccionada('ClaseManejadora');
      // Obtenemos los datos que permitirán definir el polígono y punto en el
      mapa
      $sql = "SELECT AR_DELIMITACION AS 'DELIMITACION', AR_UTMN AS 'UTMN',
      AR_UTME AS 'UTME' FROM ARQUEOLOGICOS WHERE ID={$tupla['ID']}";
      $res = $this->consultar($sql);
      $wktPoligono = $res[0]['DELIMITACION'];
      $UTMN = $res[0]['UTMN'];
      $UTME = $res[0]['UTME'];
      // Invocamos la clase gvHidraMapConfigGenerator para configurar los datos
      para el mapa.
      // Esta clase nos devolverá un JSON con los datos de configuración.
      $gen = new gvHidraMapConfigGenerator();
      // Establecemos el contenido de la capa que será la base
      $gen->setBaseLayer("WMS Inspire", "http://www.ign.es/wms-inspire/ign-
      base", "IGNBaseTodo");
      // Establecemos el polígono a mostrar
      if(!empty($tupla)) {
        $gen->setGeometry($wktPoligono, 'EPSG:25830');
      }
  }
}
```

```
// Establecemos el punto a mostrar al que se le pasará la longitud,
latitud y proyección, para poder visualizar tanto punto como polígono en el
mapa, los dos a la vez.
$gen->setPoint($UTMN, $UTME, "EPSG:25830");
// Opcional: si se quieren añadir capas con información extra.
$gen->addLayer("Cat", "http://ovc.catastro.meh.es/Cartografia/WMS/
ServidorWMS.aspx", "Catastro", 0.35);
$gen->addLayer("WMS - Uso suelo", "http://www.ign.es/wms-inspire/ign-
base", "LU.ExistingLandUse", 0.35);
// Obtenemos el JSON con la información a mostrar en el mapa
$jsonToSend = $gen->toJSON();
ob_clean();
print $jsonToSend;
die;
break;
default:
break;
}
}
```

Lista de plugins [221]

A.1.21. 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
arrayMenu	A	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 \$smarty_cadenaMenu que se sustituirá en IgepPantalla.

Ejemplo de uso del plugin cwmenulayer:

```
{cwventana tipoAviso=$smarty_tipoAviso codAviso=$smarty_codError descBreve=
$smarty_descBreve textoAviso=$smarty_textoAviso onLoad=$smarty_jsOnLoad}
{cwbarra info=$smarty_info iconOut="glyphicon glyphicon-log-out"
iconHome="glyphicon glyphicon-home" iconInfo="glyphicon glyphicon-info-
sign"}
{cwmenulayer arrayMenu="$smarty_arrayMenu"}
{/cwbarra}
{cwmarcopanel}
...
```

```
{ /cwmarcopanel }
{ /cwventana }
```

Lista de plugins [221]

A.1.22. cwpaginador

Integra una línea de enlaces para paginar cuando se presentan múltiples registros 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
conTotal-Registros	A	true	Permite controlar cuándo y dónde se visualiza el n.º total de registros en el paginador, admitiendo los siguientes valores: <ul style="list-style-type: none"> • <i>always</i>: mostrará siempre el n.º total de registros en el bloque al que afecte. • <i>never</i>: no mostrará nunca el n.º total de registros en el bloque al que afecte. • <i>only-multipage</i>: solo mostrará el n.º total cuando haya más de una página de datos. • <i>only-singlepage</i>: solo mostrará el n.º total cuando haya una única página de datos. En caso de no aparecer el argumento <i>conTotalRegistros</i> , se usa internamente por defecto el valor only-multipage . Sólo se visualizará cuando haya más de una página de datos.
enlaces-Visibles	N	true	Indica el número de enlaces que aparecieran para realizar la navegación además de los fijos (siguiente, ultimo, anterior y primero)
pagInicial	N	true	Indica la página que aparecerá visible cuando se cargue la pantalla, por defecto siempre vale 0.

Ejemplos de uso del plugin cwpaginador:

Ejemplo: Utilización de un cwpaginador en una tabla.

```
{cwtabla conCheck="true" seleccionUnica="true" datos=$smt_y_datosTabla}
{cwfila tipoListado="false"}
{cwcampotexto nombre="lisCif" size="9" editable="true" label="CIF"
dataType=$smt_y_dataType_Personas.lisCif}
{cwcampotexto nombre="lisNombre" editable="true" size="30" label="Nombre"
dataType=$smt_y_dataType_Personas.lisNombre}
{/cwfila}
```

```
{cwpaginador enlacesVisibles="3"}
{/cwtabla}
```

Lista de plugins [221]

A.1.23. 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 envíos 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
accion	A	true	Estado en el que se quiere encontrar el panel al cargarlo (modo inserción, modificación, borrado)	Puede tomar los siguientes valores: <ul style="list-style-type: none"> • <i>insertar</i> • <i>modificar</i> • <i>borrar</i>
action	A	true	Tipo de acción que debe coincidir con la correspondiente en el mappings.	
estado	A	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: <ul style="list-style-type: none"> • <i>on</i>: visible y activo • <i>off</i>: visible e inactivo • <i>inactivo</i>: no visible e inactivo
claseMa- nejadora	A	true	Indica la clase que va a ocuparse de la lógica de esta pantalla.	
detalleDe	A	true	Obligatorio cuando el panel es un detalle, en él se indicará el tipo de panel del maestro: 'lis'/'edi'.	
esMaestro	B	true	Obligatorio cuando el panel es un maestro, en ese caso valdrá "true".	

Nombre	Tipo	Opcional	Descripción	Valores
id	E	false	Fija el identificador del componente y del formulario HTML que incluye por debajo. Es necesario que dicho nombre sea único para evitar comportamientos no previstos en la interfaz.	<p>Puede tomar los siguientes valores:</p> <ul style="list-style-type: none"> • <i>fil</i>: panel de búsqueda. • <i>edi</i>: panel registro. • <i>lis</i>: panel tabulares. • <i>ediMaestro</i> / <i>ediDetalle</i>: panel registro maestro o detalle, respectivamente. • <i>lisMaestro</i> / <i>lisDetalle</i>: panel tabular maestro o detalle, respectivamente.

Ejemplos de uso del plugin cwpantallaentrada:

Ejemplo de uso del action:

```
-> Parámetro del plugin
{cwpantallaentrada 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 cwpantallaentrada en un panel de búsqueda.

```
{cwpantallaentrada id="fil" action="buscar" estado="$estado_fil"
  claseManejadora="TinvDonantes"}
{cwbarrasuppanel titulo="Donantes"}
  {cwbotontooltip iconCSS="glyphicon glyphicon-plus" label="Insertar registro"
    accion="insertar" actuaSobre="ficha" action="nuevo"}
  {cwbotontooltip iconCSS="glyphicon glyphicon-edit" label="Modificar
    registro" accion="modificar" actuaSobre="ficha"}
{/cwbarrasuppanel}
{cwcontenedor}
  {cwficha}
    {cwcampotexto nombre="filCif" editable="true" size="13" label="CIF/NIF"
      dataType=$dataType_TinvDonantes.filCif}
  {/cwcontenedor}
{cwbarrainfpanel}
  {cwbotontooltip iconCSS="glyphicon glyphicon-search" label="Buscar"
    class="boton" accion="buscar"}
{/cwbarrainfpanel}
{/cwpantallaentrada}
```

Lista de plugins [221]

A.1.24. 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
- **Pluggins que puede contener cwpantallaentrada**
 - Hijos
 - El plugin cwpantallaentrada es una hoja (no contiene otros pluggins)

Tabla de argumentos de cwpantallaentrada

Nombre	Tipo	Opcional	Descripción
codApl	A	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)
gvhidra-version	N	false	Corresponderá con la versión del framework gvHIDRA.
nomApl	A	false	Se utiliza para mostrar el nombre de la aplicación que se va a utilizar.
ubicacion	A	false	Array con las ubicaciones de las opciones en la jerarquía.
usuario	A	false	Se utiliza para mostrar en pantalla el usuario que está conectado a la aplicación.
version	N	false	Corresponderá con la versión de la aplicación.

Ejemplos de uso del plugin cwpantallaentrada:

Ejemplo: Declaración de una Pantalla de inicio. Sólo aparece en aplicación.tpl, plantilla exclusiva de igep.

```
{cwventana tipoAviso=$smtty_tipoAviso titulo=$smtty_tituloApl codAviso=$smtty_codError descBreve=$smtty_descBreve textoAviso=$smtty_textoAviso onLoad=$smtty_jsOnLoad onUnload=$smtty_jsOnUnload}
  {cwpantallaentrada usuario=$smtty_usuario nomApl=$smtty_aplicacion codApl=$smtty_codaplic version=$smtty_version gvhidraversion=$smtty_gvHidraVersion ubicacion=$smtty_ubicacion}
{/cwventana}
```

Lista de pluggins [221]

A.1.25. 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.

- **Pluggins que pueden contener a cwrichareatexto**
 - Padres
 - cwcontenedor
 - cwficha
 - cwsolapa
- **Pluggins que puede contener cwrichareatexto**
 - Hijos
 - El plugin cwrichareatexto es una hoja (no contiene otros pluggins)

Tabla de argumentos de cwrichareatexto

Nombre	Tipo	Opcional	Descripción
actualizaA	A	true	Nombre de otro componente que su valor depende del valor que tenga nuestro componte texto.
class	A	true	Corresponderá con el nombre de un estilo que se quiera aplicar de forma particular al elemento.
css	A	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.
dataType	M	false	Matriz con una estructura definida en la clase del panel. Definirá las propiedades del área de texto enriquecido, como por ejemplo obligatoriedad. Será una variable smarty en la tpl, definida de la siguiente forma: dataType = \$dataType_claseManejadora.NombreCampo
editable	A	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.
height	N	true	Especificará el alto del componente área de texto enriquecido.
label	A	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 campos obligatorios, etc...
nombre	A	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	B	true	Con "true/false" indicaremos si es un campo obligatorio a la hora de completar los datos.
showLabel	B	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 es utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.
tabIndex	N	true	Especifica el orden de tabulación.
value	A	true	Valor por defecto que apareciera en el campo.
visible	B	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.
width	N	true	Especificará el ancho del componente área de texto enriquecido.

Ejemplos de uso del plugin cwrichareatexto:

Ejemplo: Declaración de cwrichareatexto.

```
{cwrichareatexto nombre="comentario" label="Expedientes"
css="styleComentario" }
```

Lista de plugins [221]

A.1.26. 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
images	array	false	Array de imágenes que aparecerán en el visor slider.
label	A	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 campos obligatorios, etc...
nombre	A	false	Fija el identificador del componente. Es necesario que dicho nombre sea único para evitar comportamientos no previstos en la interfaz.
showLabel	B	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 es utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.

Ejemplos de uso del plugin cwslider:

Ejemplo: Declaración de un visor slider en una ficha.

```
{cwpanel id="edi" action="operarBD" estado="$estado_edi"
  claseManejadora="RegistroSlider" }
  {cwcontenedor}
    {cwfichaedicion datos=$smtty_datosFicha}
      {cwficha}
        {cwslider nombre="myImages" images=$defaultData_RegistroSlider.myImages}
      {/cwficha}
    {/cwfichaedicion}
  {/cwcontenedor}
{/cwpanel}
```

Lista de plugins [221]

A.1.27. 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
id	A	false	Es un atributo obligatorio, ya que será el identificador de la solapa.
posc-ionSolapa	N	false	Es un atributo obligatorio, que indica la posicion que tendra esa solapa. Debe ser consecutivo y empezar por 0.
title	A	false	Establace el texto que aparecerá en la solapa.

Ejemplos de uso del plugin cwsolapa:

```
{cwfichaedicion datos=$smtty_datosFicha}
{cwficha}
  {cwsolapa titulo=" Datos 1" posicionSolapa=0}
    {cwcampotexto nombre="ediCif" editable="true" size="13" label="CIF"
dataType=$dataType_Registro.ediCif}
    {cwcampotexto nombre="ediOrden" editable="true" size="2" label="Orden"
dataType=$dataType_Registro.ediOrden}
  {/cwsolapa}
  {cwsolapa titulo="Datos personales" posicionSolapa="1"}
  ...
{/cwsolapa}
{/cwficha}
{/cwfichaedicion}
```

Lista de plugins [221]

A.1.28. cwtabla

Equivalente al TABLE del HTML

• **Pluggins que pueden contener a cwtabla**

- Padres
 - cwcontenedor

• **Pluggins que puede contener cwtabla**

- Hijos
 - cwfila

Tabla de argumentos de cwtabla

Nombre	Tipo	Opcional	Descripción
conCheck	B	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.
conCheck-Todos	B	true	Si aparece y su valor es <i>true</i> , aparecera un checkbox en la cabecera de la tabla que nos permitira seleccionar y deseleccionar todos los registros. En cambio, si su valor es <i>multiSelect</i> junto al checkbox de selección de todas las tuplas de la tabla aparecera un desplegable con opciones de selección múltiple sobre los registros. Las opciones del desplegable serán las siguientes: <ul style="list-style-type: none"> • <i>Todas</i>: Seleccionará todas las tuplas de todas las páginas.

Nombre	Tipo	Opcional	Descripción
			<ul style="list-style-type: none"> • <i>Ninguna</i>: Deseleccionará todas las tuplas de todas las páginas. • <i>Página actual</i>: Añadirá a la selección actual las tuplas de la página actual. • <i>Invertir selección</i>: Invertirá la selección actual, es decir, deseleccionará las tuplas seleccionadas, y seleccionará las tuplas que no estén seleccionadas.
conTotalRegistros	A	true	<p>Permite controlar cuándo se visualiza el n.º total de registros al pie de la tabla, admitiendo los siguientes valores:</p> <ul style="list-style-type: none"> • <i>always</i>: mostrará siempre el n.º total de registros en el bloque al que afecte. • <i>never</i>: no mostrará nunca el n.º total de registros en el bloque al que afecte. • <i>only-multipage</i>: solo mostrará el n.º total cuando haya más de una página de datos. • <i>only-singlepage</i>: solo mostrará el n.º total cuando haya una única página de datos. <p>En caso de no aparecer el argumento <i>conTotalRegistros</i>, se usa internamente por defecto el valor never. Solamente se visualizará cuando haya más de una página de datos.</p>
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 (<i>\$smtty_datos</i>) que se sustituye internamente.
numFilasPantalla	N	true	Fija el número de filas de datos que queremos que aparezca por pantalla. En caso de no aparecer por defecto aparecerán 6 filas de datos.
seleccionUnica	B	true	Permite un único elemento seleccionado en la tabla.

Ejemplos de uso del plugin cwtabla:

```
{cwtabla conCheck="true" seleccionUnica="true" datos=$smtty_datosTabla}
{cwfila tipoListado="false"}
  {cwcampotexto nombre="lisCif" size="9" editable="true" label="CIF" dataType=
  $dataType_MiClase.lisCif}
  {cwcampotexto nombre="lisNombre" editable="true" size="30" label="Nombre"
  dataType=$dataType_MiClase.lisNombre}
  {cwcampotexto nombre="lisMoto" editable="true" size="30" label="Moto"
  dataType=$dataType_MiClase.lisMoto}
{/cwfila}
{cwpaginador enlacesVisibles="3"}
{/cwtabla}
```

Lista de plugins [221]

A.1.29. cwupload

Equivale al FILE de HTML

- **Plugins que pueden contener a cwupload**
 - Padres
 - El plugin cwupload es raíz (no lo contiene ningún otro plugin)

- **Plugins que puede contener cwupload**

- Hijos
 - cwficha
 - CWConenedor

Tabla de argumentos de cwupload

Nombre	Tipo	Opcional	Descripción
class	A	false	Se podrá especificar un estilo particular para el componente.
editable	B	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.
icono	A	true	Icono (CSS) que aparecerá en el botón del upload.
label	A	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.
multiple	B	false	Con este parámetro se indicará si se permite la subida de más de un fichero.
nombre	A	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	B	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.
showLabel	B	true	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.
tabIndex	N	true	Especifica el orden de tabulación.

Ejemplos de uso del plugin cwupload:

```
{cwupload nombre="ficheroUpload" size="10" label="Si quieres cambiar la imagen..."}

```

Lista de plugins [221]

A.1.30. 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

Tabla de argumentos de cwventana

Nombre	Tipo	Opcional	Descripción	Valores
codAviso	A	false	Fija el código de aviso. Variable smarty \$smarty_codError.	
descBreve	A	false	Descripción breve del mensaje del aviso. Variable smarty \$smarty_descBreve.	
layout	A	false	Contendrá unos estilos con los que se podrá elegir como disponer el menú y la botonera ('búsqueda', 'listado', 'edición').	Valores posibles: <ul style="list-style-type: none"> • <i>vertical-menu</i>: Con esta opción el menú aparecerá de forma vertical en el lado izquierdo de la pantalla. Se ocultará cada vez que se seleccione una opción. • <i>vertical-menu-visible</i>: Añadiendo esta opción el menú no se ocultará, siempre estará fijo en el lateral. • <i>botonera-tabs</i>: La botonera aparecerá de forma vertical en el lado derecho de la pantalla.
manageSession	B	false	Con este parámetro indicaremos si se quiere mantener la sesión activa o no para la ventana.	
onload	A	false	En este parámetro podremos introducir una llamada a una función javascript que debe ejecutarse en el evento onLoad de la página	
onUnload	A	false	En este parámetro podremos introducir una llamada a una función javascript que debe ejecutarse en el evento onUnLoad de la página	
textoAviso	A	false	Descripción detallada del aviso. Variable smarty \$smarty_textoAviso.	
tipoAviso	A	false	Indica el tipo de aviso según la guía de estilo. Variable smarty \$smarty_tipoAviso.	Los tipos de aviso pueden ser: <ul style="list-style-type: none"> • <i>aviso</i> • <i>alerta</i> • <i>notificación</i> • <i>sugerencia</i>
title	A	true	Fija el título de la Ventana HTML. Variable smarty \$smarty_tituloApl.	

Ejemplos de uso del plugin cwventana:


```
{cwventana tipoAviso=$smtyp_tipoAviso codAviso=$smtyp_codError descBreve =  
  $smtyp_descBreve textoAviso=$smtyp_textoAviso onLoad=$smtyp_jsOnLoad}  
...  
{/cwventana}
```

Lista de plugins [221]

Apéndice B. Listados Jasper en gvHIDRA

B.1. Integración 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 incluyamos en el fichero .cvsignore del proyecto una línea con la palabra "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 subdirectorío 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 producción 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 acción 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('martel');
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.
- Líneas 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, que en PHP son objetos).
 - Valor del parámetro: enviaremos el valor (ojo con tipos complejos, como las fechas, que 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. Aunque 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. Opcionalmente, 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
2: require_once('include/jasper/modulosPHP/informeJasper.php');
3:
4:
5:     if (IgepSession::existeVariable("claseDemo", "infJasper_demo"))
6:     {
7:         $informeJ = &
IgepSession::dameVariable("claseDemo", "infJasper_demo");
8:         $informeJ->createResultFile('pdf');
9:     }
10:     IgepSession::borraVariable("claseDemo", "lanzarInforme");
11:
12:
...SIGUE...
```

Del código anterior destacan:

- Línea 10: Recuperamos el atributo que referencia al listado de la sesión.
- Línea 11: Invocamos la creación del informe.

B.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 línea 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 librerías necesarias.
- comando que solicita la ejecución de la librería 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 envíen 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 obtendríamos 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(JRJdbcQueryE  
at  
net.sf.jasperreports.engine.fill.JRFillDataset.createQueryDatasource(JRFillDataset.  
at  
net.sf.jasperreports.engine.fill.JRFillDataset.initDatasource(JRFillDataset.java:614  
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:402)  
at  
net.sf.jasperreports.engine.JasperFillManager.fillReport(JasperFillManager.java:234)  
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(SimpleParameterList
  at
org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:182)
  at
org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:452)
  at
org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statement.
  at
org.postgresql.jdbc2.AbstractJdbc2Statement.executeQuery(AbstractJdbc2Statement.java
  at
net.sf.jasperreports.engine.query.JRJdbcQueryExecuter.createDatasource(JRJdbcQueryEx
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 línea más importante de todo el listado ya que en ella JasperReports intentará explicarnos la causa del error.

Seguiremos intentado arreglar 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 línea define('DEBUG', FALSE);

B.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: \${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 cláusula 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 algún 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 C. Pasos de migración para pasar a la versión 5.0.0

1. Codificación del proyecto. [266]
2. Cambios en la lógica de negocio. [266]
3. Migración de los ficheros XML de configuración. [267]
4. Migración del fichero AppMainWindow.php [268]
5. Migración del fichero mappings.php [268]
6. Migración de las plantillas (TPL). [268]
7. Migración de los views. [275]

C.1. Codificación del proyecto.

A partir de la versión 5.0.0 la codificación de los proyectos gvHIDRA será UTF-8, por lo tanto, el primer paso a realizar es la recodificación de nuestra aplicación.

C.2. Cambios en la lógica de negocio.

- **Conexión a la BD**

La conexión a la BD penaliza bastante el rendimiento, por eso se ha realizado un cambio para retardar el proceso de conexión hasta el momento necesario. Esto afectará a todo el código fuente que haga uso de la clase *IgepConexion*.

El paso de migración consiste en buscar la cadena *new IgepConexion* y en todas aquellas líneas *incorporar* la llamada al método *conectar*:

```
$conexion = new IgepConexion ($dsn);
$conexion->conectar();
```

- **Migración de las listas de clase**

En el caso de las listas definidas en clase se debe pasar la conexión como parámetro en el constructor. Por lo tanto el constructor quedará de la siguiente forma:

```
protected $_conDB;
protected $_lang;

public function __construct ($conn=null) {
    if (is_object ($conn)) {
        $this->_conDB = $conn;
    } else {
        try {
            $conf = ConfigFramework::getConfig();
            $this->_lang = $conf->getLanguage();
            $dsn = $conf->getDSN ('g_dsn');
            $this->_conDB = new IgepConexion ($dsn);
        } catch (Exception $e) {
```

```
IgepDebug::setDebug (ERROR, 'Error en clase ' . __CLASS__ . ':\n'
    . PHP_EOL . $e->getMessage());
throw new gvHidraException ('Error al conectar a la BD. '
    . 'Contacta con el administrador de la aplicación.');
```

- **Revisión de paso de parámetros por \$_GET**

Se ha cambiado el sistema de comunicación a jQuery + JSON. La información que compone las matrices de datos se obtiene a través de una matriz de datos que se mantiene en JavaScript. Esto es transparente al programador excepto para **aquellas llamadas que utilizaban la url para pasar parámetros**, esto ya **NO** funcionará.

- **Nueva matriz de datos para los datos del panel FIL**

Principalmente afecta a listados generados a partir de un panel de búsqueda. Hasta ahora, las acciones particulares que hacían uso de los datos de un panel de búsqueda, hacían referencia a la matriz de datos *external* para localizarlos. Para evitar confusiones en esta versión, estos datos van a estar almacenados en la matriz de "visibles". Esto significa que se deberán revisar estas acciones (típicamente en listados) y cambiar el método *setOperation*

```
$objDatos->setOperation ('visibles');
```

- **Constructores de listas de clase y ventanas de selección**

Cuando se utilizan clases para definir listas o ventanas de selección el constructor pasa a tener un parámetro relativo a la conexión. De modo que la definición del constructor quedará de la siguiente forma:

```
protected $_conDB;
protected $_lang;

public function __construct ($conn=null) {
    if (is_object ($conn)) {
        $this->_conDB = $conn;
    } else {
        try {
            $conf = ConfigFramework::getConfig();
            $this->_lang = $conf->getLanguage();
            $dsn = $conf->getDSN ('g_dsn');
            $this->_conDB = new IgepConexion ($dsn);
        } catch (Exception $e) {
            IgepDebug::setDebug (ERROR, 'Error en clase ' . __CLASS__ . ':\n'
                . PHP_EOL . $e->getMessage());
            throw new gvHidraException ('Error al conectar a la BD. '
                . 'Contacta con el administrador de la aplicación.');
```

C.3. Migración de los ficheros XML de configuración.

Se extrae la DTD de los ficheros xml de configuración, quedando así más limpios y sin necesidad de tener que ir migrando las DTD's conforme evoluciona el framework. Por lo tanto, hay que eliminar la DTD incluida en los ficheros XML, y añadirla de forma relativa tal y como se indica a continuación:

- **gvHidraConfig.inc.xml** Eliminar la DTD y añadir lo siguiente:

```
<!DOCTYPE gvHidraConfig SYSTEM "igep/dtd/configAPP.dtd">
```

- **menuModulos.xml** Eliminar la DTD y añadir lo siguiente:

```
<!DOCTYPE menu SYSTEM "../igep/dtd/menu.dtd">
```

- **menuHerramientas.inc.xml** Eliminar la DTD y añadir lo siguiente:

```
<!DOCTYPE menu SYSTEM "../igep/dtd/menu.dtd">
```

- **menuAdministracion.inc.xml** Eliminar la DTD y añadir lo siguiente:

```
<!DOCTYPE menu SYSTEM "../igep/dtd/menu.dtd">
```

C.4. Migración del fichero AppMainWindow.php

Solamente afecta a la función **emptyLogTable()**

```
public function emptyLogTable ($dias=60) {
    //Recogemos dsn de conexion
    $conf = ConfigFramework::getConfig();
    $g_dsnLog = $conf->getDSN ('gvh_dsn_log');
    $usuario = IgepSession::dameUsuario();
    try {
        IgepDebug::purgeDBLog ($dias, $usuario, $g_dsnLog);
    } catch (Exception $e) {
        error_log (__FILE__ . __METHOD__ . ' Error al vaciar tabla de LOG');
    }
    return 0;
} //emptyLogTable
```

C.5. Migración del fichero mappings.php

Si el botón **accion="cancelar"** no tiene definido el parámetro **action** en la plantilla *tpl*, hay que asegurarse que la regla *mapping* está definida la acción **cancelarEdicion**.

```
$this->_AddMapping ('ClaseM__cancelarEdicion','ClaseM');
$this->_AddForward ('ClaseM__cancelarEdicion',
    'gvHidraSuccess',
    'index.php?view=views/p_ClaseM.php&panel=listar');
```

C.6. Migración de las plantillas (TPL).

Las plantillas son las que más migración requieren. Por una parte, se ha evolucionado la versión de Smarty que incorpora el framework y, por otro lado, se han creado nuevos parámetros en los plugins, cambiado otros, y eliminados algunos plugins que ya son obsoletos.

C.6.1. Plugins obsoletos.

Se deben eliminar de las plantillas *TPL* los siguientes plugins:

- {CWContenedorPestanya} ... {/CWContenedorPestanya}
- {CWPEstanya}
- {CWSelector} ... {/CWSelector}

C.6.2. Maestro/Detalle.

En las plantillas *TPL* que corresponden a un maestro-detalle hay que eliminar los tags que separaban el maestro del detalle.

```
</td></tr><tr><td>
```

C.6.3. Migración del nombre de los plugins.

Debido a la migración de Smarty en su versión 3, los plugins pasan a ser escritos en **minúsculas**, por lo tanto hay que reemplazar en todas las plantillas *tpl* el nombre de los plugins por el mismo pero en minúsculas.

Para el reemplazo se aconseja seguir el orden que se indica a continuación:

- CWPantallaEntrada --> **cwpantallaentrada**
- CWVentana --> **cwventana**
- CWBarraSupPanel --> **cwbarrasuppanel**
- CWBarraInfPanel --> **cwbarrainfpanel**
- CWBarra --> **cwbarra**
- CWMenuLayer --> **cwmenulayer**
- CWPanel --> **cwpanel**
- CWBotonTooltip --> **cwbotontooltip**
- CWBoton --> **cwboton**
- CWMarcoPanel --> **cwmarcopanel**
- CWContendor --> **cwcontendor**
- CWFichaEdicion --> **cwfichaedicion**
- CWFicha --> **cwficha**
- CWTabla --> **cwtabla**
- CWFila --> **cwfila**
- CWPaginador --> **cwpaginador**
- CWSolapa --> **cwsolapa**
- CWInfoContendor --> **cwinfocontendor**
- CWInformation --> **cwinformation**
- CWDetalles --> **cwdetalles**
- CWGraph --> **cwgraph**
- CWImagen --> **cwimagen**
- CWLabel --> **cwlabel**

- CWMaps --> **cwmaps**
- CWAreaTexto --> **cwareatexto**
- CWCampoTexto --> **cwcampotexto**
- CWCheckBox --> **cwcheckbox**
- CWLista --> **cwlista**
- CWRichAreaTexto --> **cwrichareatexto**
- CWUpload --> **cwupload**

C.6.4. Notas importantes sobre los parámetros de los plugins.

En algunos plugins se añaden parámetros que son altamente necesarios para mejorar la funcionalidad del framework.

- **{cwbotontooltip}**

Obligatorio el parámetro **id**.

- **{cwboton}**

Obligatorio el parámetro **id**.

En el caso de que el parámetro

```
accion="particular"
```

y se quiere que el botón sea visible sin necesidad de activar el panel para edición, se debe añadir el parámetro

```
visible="true"
```

- **{cwsolapa}**

Obligatorio el parámetro **id**.

- **{cwlabel}**

Obligatorio el parámetro **tipo**. Con él se indicará el tipo de etiqueta que se quiere mostrar:

- **icon**: icono
- **iconLabel**: icono + etiqueta
- **label**: etiqueta
- **link**: enlace
- **iconLink**: icono + enlace

C.6.5. Parámetros de plugins que se deben eliminar.

Se deben eliminar algunos parámetros de ciertos plugins porque no es necesario indicarlos en las plantillas *tpl* porque se pueden obtener internamente.

- **{cwbotontooltip}**

Eliminar el parámetro: **imagen** (Si aún se tiene este parámetro porque viene de la rama 4.2.x)

- **{cwboton}**

Eliminar el parámetro: **imagen** (Si aún se tiene este parámetro porque viene de la rama 4.2.x)

- **{cwfichaedicion}**

Eliminar el parámetro: **id**

- **{cwmarcopanel}**

conPestanyas parámetro opcional con el que se puede controlar si se visualizan o no la botonera ['nueva búsqueda','listado','edición']. Por defecto su valor es "true", se visualiza.

- **{cwpanel} ... {/cwpanel}**

Eliminar los parámetros: **method** y **tipoComprobacion**

- **{cwtabla}**

Eliminar el parámetro: **id**

C.6.6. Parámetros de plugins que se deben renombrar.

Se renombran algunos parámetros para unificar criterios.

- **{cwbarra}**

Los siguientes parámetros:

```
usuario=$smt_y_usuario codigo=$smt_y_codigo customtitle=$smt_y_customTitle
```

Se unifican en uno solo que será un array, por lo que deberá ser sustituido por el siguiente parámetro:

```
info=$smt_y_info
```

- **{cwbarrasuppanel}**

El parámetro *titulo* renombrarlo por *title*.

- **{cwbotontooltip}**

- El parámetro *funcion* renombrarlo por *accion*.
- El parámetro *titulo* renombrarlo por *title*.

- **{cwboton}**

El parámetro *texto* renombrarlo por *label*.

- **{cwareatexto}, {cwcampotexto}, {cwcheckbox}, {cwlista}, {cwrichareatexto}, {cwupload}, {cwimagen}, {cwlabel}, {cwslider}, {cwinformation}**

- El parámetro *textoAsociado* renombrarlo por *label*.
- El parámetro *mostrarTextoAsociado* renombrarlo por *showLabel*.

- **{cwmenulayer}**

El parámetro *cadenaMenu*="\$smt_y_cadenaMenu" renombrarlo por *arrayMenu*=\$smt_y_arrayMenu

- **{cwsolapa}**

El parámetro *titulo* renombrarlo por *title*

C.6.7. Paso de la variable \$smtty_iteracionActual al uso de la librería IgepJS.js

En esta nueva versión del framework la comunicación cliente-servidor ha cambiado, ahora no se tienen todas las páginas cargadas en el cliente por lo que cada cambio de página es una actualización del contenido de los campo sin recarga de la página completa. Este cambio implica que la variable \$smtty_iteracionActual que se utilizaba para adaptar la interfaz según ciertos criterios, ya no es posible utilizarla porque solamente se tendrá cargado un registro cada vez (una iteración).

Para no perder esta funcionalidad se ha creado una librería javascript, **gvh_IgepJS.js**, donde se encuentra una serie de funciones relacionadas con la interfaz y que permitirán realizar la adaptación de la misma. La actualización de la interfaz ahora se adaptará cuando cambie el contenido del campo. Es decir si un campo, del que depende una modificación de interfaz, altera su valor lanzará una llamada al método correspondiente de gvh_IgepJS y se actualizará la interfaz.

C.6.7.1. Funciones incluidas en la librería.

- ```
/**
 * getStatePanel: Obtenemos el estado del panel ('R', 'W', 'I')
 *
 * @access public
 * @return string Estado del panel ('R', 'W', 'I').
 */
gvh.IgepJS.prototype.getStatePanel = function()
```
- ```
/**
 * _panelVisible: Devuelve si el panel está visible o no
 *
 * @access private
 * @param panel: id del panel a consultar
 */
gvh.IgepJS.prototype._panelVisible = function()
```
- ```
/**
 * getRowComponent: Obtiene el row del componente indicado
 *
 * @access public
 * @param componente: Nombre del campo
 * @return integer
 */
gvh.IgepJS.prototype.getRowComponent = function(componente)
```
- ```
/**
 * getIdComponent: Obtiene el id del componente indicado
 *
 * @access public
 * @param {string} componente - Nombre del campo
 * @param {boolean} external - [true / false]
 * @param {integer} [row] - Fila del componente
 */
gvh.IgepJS.prototype.getIdComponent = function( componente, external, row )
```
- ```
/**
 * getValue: Obtiene el valor del campo
 *
```

```

 * @access public
 * @param componente: Nombre del campo
 * @param external: [true / false]
 */
 gvh.IgepJS.prototype.getValue = function(componente, external)

• /**
 * setValue: Asigna valor a un campo
 *
 * @access public
 * @param componente: Nombre del campo
 * @let valor: Valor a asignar al componente
 * @param external: [true / false]
 */
 gvh.IgepJS.prototype.setValue = function(componente, valor, external)

• /**
 * getClass: Obtiene el class del campo indicado
 *
 * @access public
 * @param componente: Nombre del campo
 * @param external: [true / false]
 */
 gvh.IgepJS.prototype.getClass = function(componente, external)

• /**
 * addClassComponent: Modifica el class de un componente
 *
 * @access public
 * @param componente: Nombre del campo
 * @param css: estilo css a añadir
 * @param external: [true / false]
 */
 gvh.IgepJS.prototype.addClassComponent = function(componente, css,
 external)

• /**
 * removeClassComponent: Modifica el class de un componente
 *
 * @access public
 * @param componente: Nombre del campo
 * @param css: estilo css a añadir
 * @param external: [true / false]
 */
 gvh.IgepJS.prototype.removeClassComponent = function(componente, css,
 external)

• /**
 * getVisible: Obtener si un campo es visible o no
 *
 * @access public
 * @param componente: Nombre del campo
 * @param external: [true / false]
 */
 gvh.IgepJS.prototype.getVisible = function(componente, external)

• /**
 * setVisible: Mostrar/ocultar un campo
 *

```



```

* @access public
* @param componente: Nombre del campo
* @param visible: [true / false]
* @param external: [true / false]
*/
gvh.IgepJS.prototype.setVisible = function(componente, visible, external)

• /**
* getEnable: Obtener si un campo está habilitado o no
*
* @access public
* @param componente: Nombre del campo
* @param external: [true / false]
*/
gvh.IgepJS.prototype.getEnable = function(componente, external)

• /**
* setEnable: Habilitar/deshabilitar un campo
*
* @access public
* @param componente: Nombre del campo
* @param enable: [true / false / nuevo]
* @param external: [true / false]
*/
gvh.IgepJS.prototype.setEnable = function(componente, enable, external)

```

### C.6.7.2. Uso de la librería

Como se ha indicado, la actualización de interfaz se realiza con JavaScript por lo tanto la programación se debe incluir en la tpl como `<script></script>` con el atributo **defer** porque interesa que se ejecute lo último, cuando ya esté todo cargado, al igual que nos aseguramos que jQuery también está cargado. En el caso de que la actualización de la interfaz corresponda a un panel detalle colocar el bloque `<script>...</script>` antes de cerrar el `{/if}` del detalle.

```

{if count($smtly_datosFichaM) gt 0 }
...
<script type="text/javascript" defer>
...
</script>
{/if}

```

Primero se tiene que invocar al método `subscribeRewireUI` para tener suscrita la lógica de interfaz correspondiente a la clase manejadora del panel, y crear el objeto de la clase `IgepJS` de la clase manejadora para poder invocar las funciones de la librería.

Ejemplo de uso y correspondencia con el uso de la variable `$smtly_iteracionActual`:

Uso de `$smtly_iteracionActual`:

```

{if $smtly_datosFicha.$smtly_iteracionActual.edi_componente eq "S"}
 {CWCampoTexto nombre="ediDescripcion" size="40" editable="true" value=
 $defaultData_ClaseManejadora.ediDescripcion}
{else}
 {CWCampoTexto nombre="ediDescripcion" size="40" editable="false" value=
 $defaultData_ClaseManejadora.ediDescripcion}
{/if}
{if $smtly_datosFicha.$smtly_iteracionActual.edi_perfil eq "ADMINISTRACION"}
 {CWCampoTexto nombre="ediTelefono" size="10"
 editable="true" visible="true" value=
 $defaultData_ClaseManejadora.ediTelefono}

```

```
{else}
 {CWCampoTexto nombre="ediTelefono" size="10"
 editable="true" visible="false" value=
 $defaultData_ClaseManejadora.ediTelefono}
{/if}
```

Uso de la librería gvh\_IgepJS:

```
<script type="text/javascript" defer>
$(document).ready(function() {
 // Invocación al método subscribeRewireUI para subscribir la lógica de
 interfaz (función) a la clase manejadora del panel.
 gvh.subscribeRewireUI("ClaseManejadora",function()
 {
 // Crear un objeto de la clase IgepJS pasándole la clase manejadora y el
 tipo de panel ('lis','lisDetalle','edi','ediDetalle').
 objIgepJS_ClaseManejadora = new gvh.IgepJS('ClaseManejadora','edi');

 // Uso de los métodos de gvh_IgepJS
 if (objIgepJS_ClaseManejadora.getValue('edi_componente') == 'S')
 {
 objIgepJS.setEnabled('ediDescripcion', true, false);
 }
 else {
 objIgepJS.setEnabled('ediDescripcion', false, false);
 }

 if (objIgepJS.getValue('edi_perfil') == 'ADMINISTRACION')
 {
 objIgepJS.setVisible('ediTelefono', true, false);
 }
 else {
 objIgepJS.setVisible('ediTelefono', false, false);
 }
 }
});
</script>
```

## C.7. Migración de los views.

En el caso de pantallas tipo **Maestro-Ndetalles** hay un cambio en los **views**. Hay que añadir el tipo de panel en el array de definición de los paneles detalle.

```
$detalles = array()
array(
 "panelActivo" => "Detalle 1" ,
 "titDetalle" => "Título detalle 1" ,
 "panel" => "lis"
) ,
array(
 "panelActivo" => "Detalle 2" ,
 "titDetalle" => "Título detalle 2" ,
 "panel" => "edi"
)
```

# Apéndice D. Novedades de la versión

En este punto se destaca aspectos más relevantes que incorpora la versión.

1. **Herencia en plantillas, plugins y ficheros de idiomas.** [276]
2. **Mejora de rendimiento en la visualización de resultados.** [277]
3. **Nuevos customs.** [277]
4. **Menú de la aplicación.**
5. **Configuración aspectos generales de las ventanas.** [280]
  - a. Icono "modificado" en barra superior [280]
  - b. Ubicación botonera: Nueva búsqueda, listado, edición. [281]
  - c. Redimensionar paneles [282]
  - d. Conteo de registros en las solapas de los detalles [283]
6. **Configuración nuevas funcionalidades en paneles tabulares.** [284]
  - a. Opciones de selección de registros en un tabular [284]
  - b. Selección directamente en la fila. [285]
  - c. Mostrar número total de registros en un tabular [286]
  - d. Parámetro *column\_align* para campos en un panel tabular. [287]
7. **Nuevas operaciones y parámetros en plugins.** [287]
  - a. *cwinfocontenedor*. Operaciones *getValue()* y *setValue()* [287]
  - b. *cwinformation*. Operaciones *getValue()* y *setValue()*. Parámetros nuevos. [288]
  - c. *cwbotontooltip* con funcionamiento independiente del estado del panel. [289]
  - d. Visibilidad de etiquetas que acompañan componentes. [289]
  - e. Parámetro *confirm* en botones tooltip. [289]
8. **Debug en desarrollo.** [290]
  - a. Debug de plantillas en tiempo de ejecución. [290]
  - b. Debug de JavaScript. [290]

## D.1. Herencia en plantillas, plugins y ficheros de idiomas.

Con esta herencia se abre la posibilidad de poder particularizar ciertos comportamientos para una aplicación en concreto sin tener que depender del framework.

Se han añadido tres niveles de herencia a la hora de trabajar en GVHIDRA con plantillas, plugins y ficheros de idiomas de Smarty. De este modo, se permite tanto en **aplicación como en custom personalizar** los distintos elementos Smarty disponibles en GVHIDRA sin tener que depender de cambios en el framework, permitiendo enriquecer el comportamiento de las aplicaciones. La precedencia de la **herencia es App > Custom > Framework**, esto es,

cualquier elemento Smarty (plantilla, plugin, fichero idioma) se buscará primero en la aplicación, si no está disponible, se buscará entonces en Custom, y si tampoco está disponible se buscará entonces en el código del framework GVHIDRA.

Las ubicaciones de los distintos elementos Smarty son:

- Ficheros idiomas --> **/lang/**
- Plantillas tpl --> **/plantillas/**
- Plugins --> **/smarty/plugins/**

Por tanto, si dentro de la aplicación se crea un fichero `function.cwlista.php` dentro de la carpeta `app/smarty/plugins/`, el comportamiento del plugin `cwlista` pasaría a ser el definido por dicho fichero, ignorando el comportamiento estándar GVHIDRA que se venía usando desde `app/igep/smarty/gvhplugins/function.cwlista.php`, permitiendo así modificar/sobreescribir comportamientos sin depender del núcleo GVHIDRA, y también añadir nuevas funcionalidades mediante nuevos plugins, etc. De forma análoga se podría sobreescribir o extender las plantillas (.tpl) usadas por los plugins, y lo mismo con los ficheros de idiomas (.conf).

## D.2. Mejora de rendimiento en la visualización de resultados.

En esta versión cuando se realiza una consulta, y ésta devuelve más de un registro, no se "dibujan" todas las capas (<div>...</div>) correspondientes a cada página como se hacía antes. Hasta ahora, cuando se paginaba lo que se hacía era hacer visible la capa de la página correspondiente, ocultando la anterior. Crear tal cantidad de capas, cuando la consulta devolvía un número alto de registros, podía resultar pesado.

En esta versión se ha cambiado la comunicación entre presentación y negocio utilizando ahora llamadas Ajax, y con ello se crea una estructura JSON con toda la información del resultado de la consulta. Por lo tanto, respecto al código HTML ahora sólo se crea una página (capa <div>...</div>), teniendo toda la información del resultado de la consulta en el objeto JSON, la paginación se realiza obteniendo los valores correspondientes del objeto JSON y asignándolos a cada uno de los campos de la página dibujada.

Por lo tanto el peso de la página que se obtiene para cada pantalla ha disminuido de forma contundente, haciendo que el tiempo de acceso a cada pantalla en caso de una gran cantidad de registros ha disminuido.

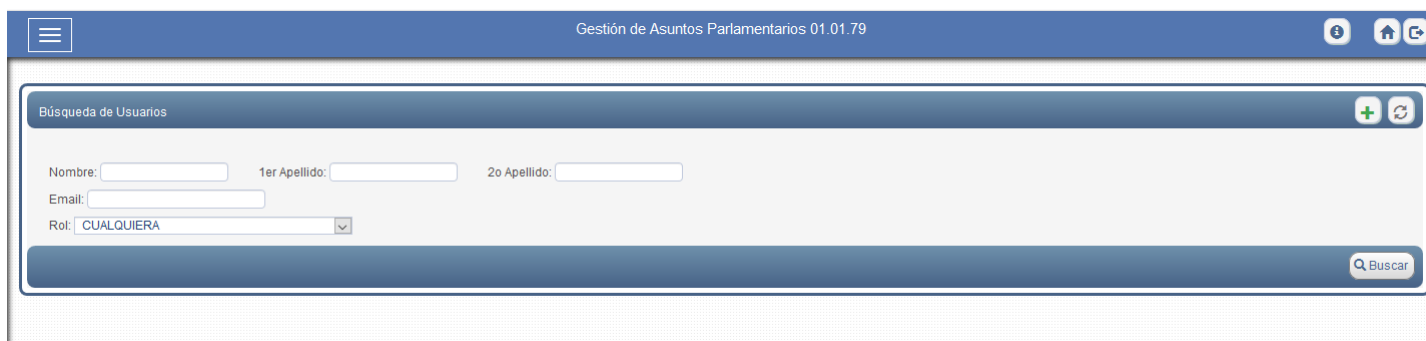
## D.3. Nuevos customs.

En el paquete que se distribuye del framework de esta versión se incluyen varios customs para poder utilizar al desarrollar aplicaciones.

- Custom **greyStyle**

Es el estilo clásico de gvHidra, el que se venía distribuyendo hasta ahora.

**Figura D.1. Custom greyStyle**



- Custom **lightStyle**

Se ha creado una versión del custom con colores más claros, elementos más grandes y espaciados entre sí. Se incluye una versión compact de este estilo: **cpLightStyle**

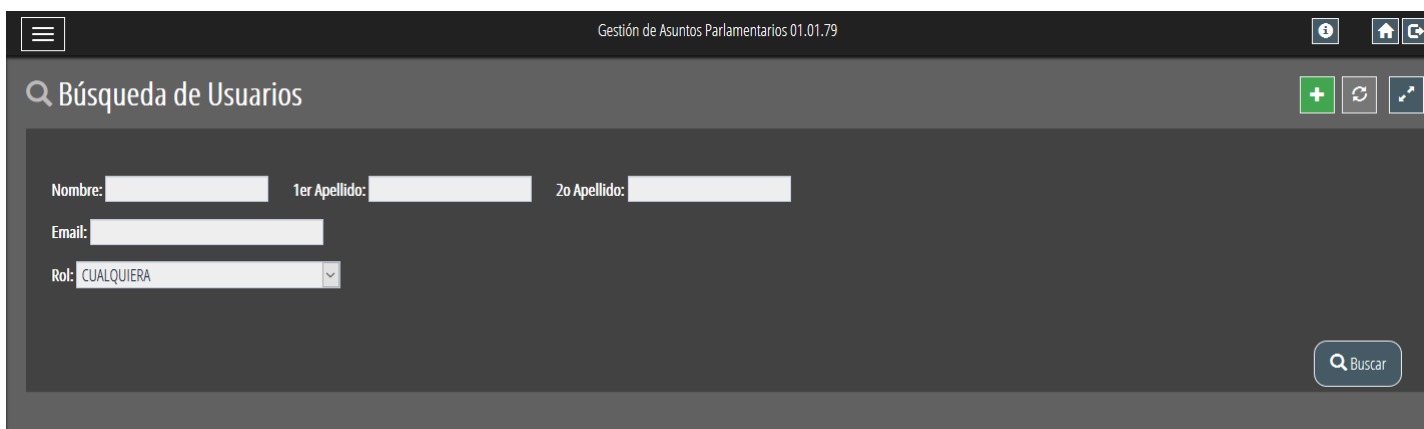
**Figura D.2. Custom lightStyle**



- Custom **darkStyle**

El darkStyle es una versión, como su propio nombre indica, más oscura del custom con tonalidades grises y negro, elementos más grandes y espaciados entre sí. Se incluye una versión compact de este estilo: **cpDarkStyle**

**Figura D.3. Custom darkStyle**



## D.4. Menú de la aplicación.

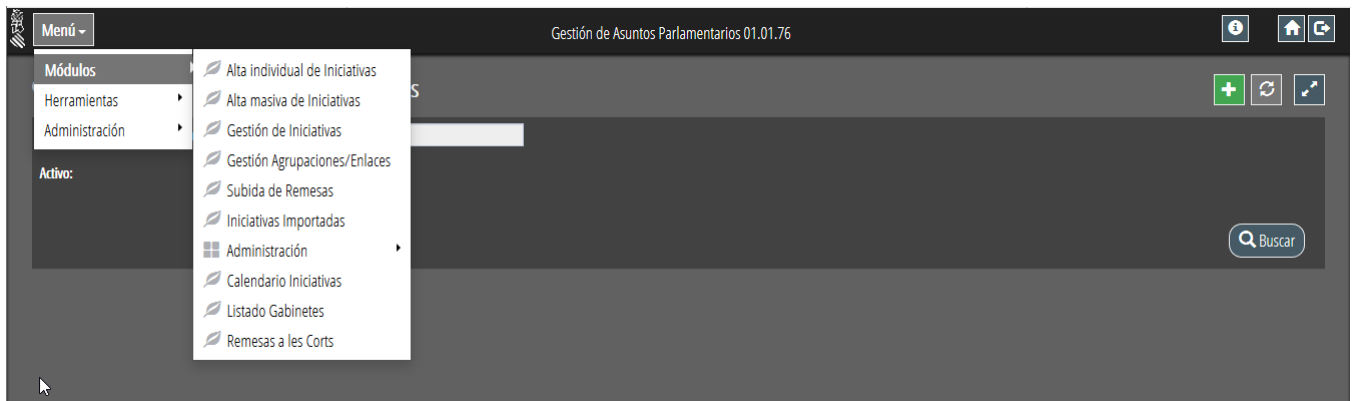
Un aspecto destacable de esta versión es la posibilidad de elegir la posición del menú. El plugin **cwventana** tiene el parámetro **layout** que puede contener una serie de valores (selectores css) que determinarán cierta visualización de algún elemento. Estos valores son los que permitirán elegir las características y posición del menú.

- **Menú en la barra superior de la ventana.**

Es la posición clásica de gvHIDRA, por lo tanto, es la opción por defecto del framework, lo que implica que no es obligatorio ponerlo. En este caso el parámetro **layout** podría contener el valor **classic-menu**.

```
{cwventana layout="classic-menu" titulo="APL"
tipoAviso=$smtyp_tipoAviso codAviso=$smtyp_codError descBreve=$smtyp_descBreve
textoAviso=$smtyp_textoAviso onLoad=$smtyp_jsOnLoad}
```

**Figura D.4. Menú barra superior (custom darkStyle).**



- **Menú en una barra lateral a la izquierda.** El menú se ubicará en una barra lateral en la parte de la izquierda de la pantalla. Inicialmente, al entrar a una pantalla con este menú aparecerá escondido, para visualizarlo se tendrá que hacer click en el botón de despliegue del menú.



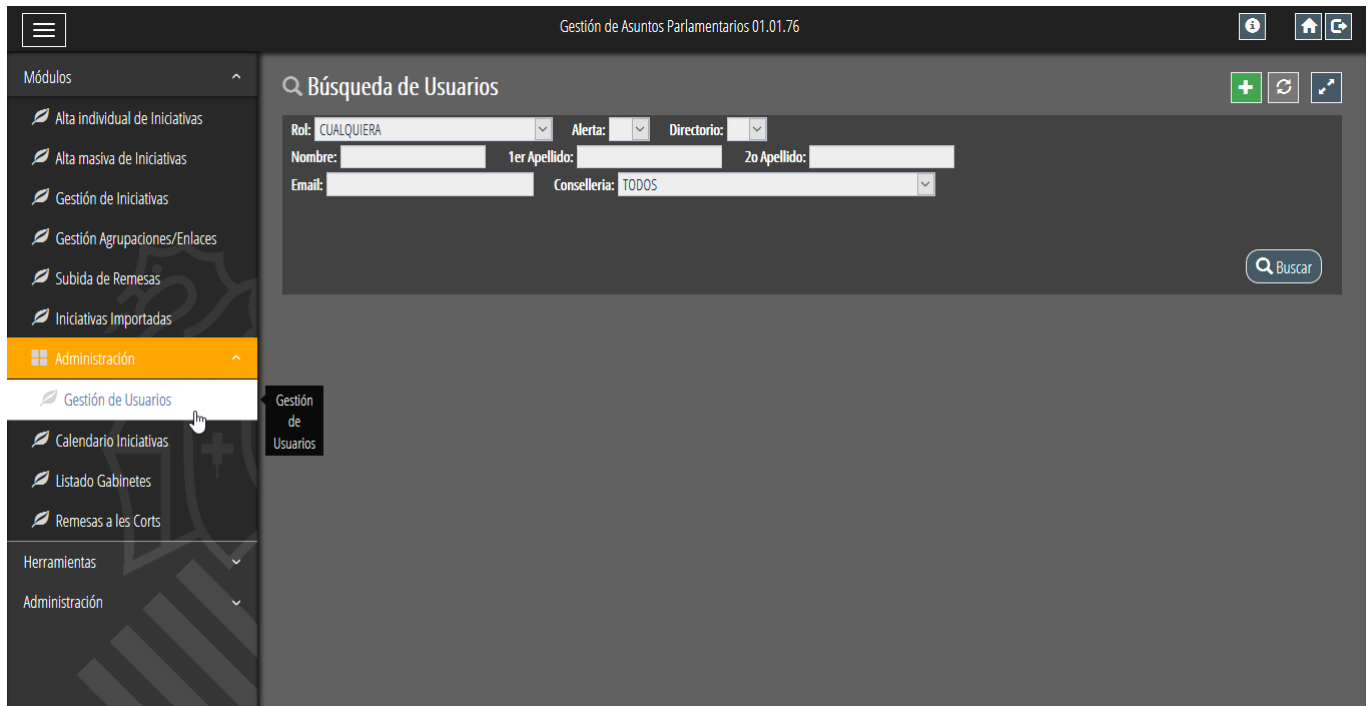
Para este funcionamiento, el parámetro **layout** deberá contener el valor **vertical-menu**.

```
{cwventana layout="vertical-menu" titulo="APL"
tipoAviso=$smtty_tipoAviso codAviso=$smtty_codError descBreve =
 $smtty_descBreve textoAviso=$smtty_textoAviso onLoad=$smtty_jsOnLoad}
```

En el caso de que interese mantener el menú de forma fija en el lado izquierdo, que no se esconda cada vez que se pulse en una opción, al parámetro **layout** se le debe añadir el valor **vertical-menu-visible**.

```
{cwventana layout="vertical-menu vertical-menu-visible" titulo="APL"
tipoAviso=$smtty_tipoAviso codAviso=$smtty_codError descBreve =
 $smtty_descBreve textoAviso=$smtty_textoAviso onLoad=$smtty_jsOnLoad}
```

Figura D.5. Menú barra lateral (custom darkStyle).



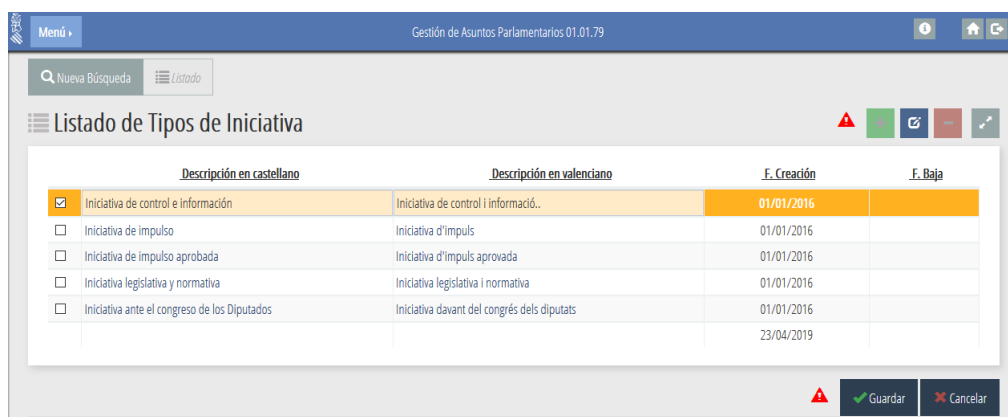
## D.5. Configuración aspectos generales de las ventanas.

### D.5.1. Icono "modificado" en barra superior.

Tanto en un panel tabular como en una ficha, cuando se está modificando algún dato o insertando un registro nuevo aparece en la barra inferior un icono que indica que el panel contiene modificaciones. En el caso de que el panel sea más extenso que el tamaño del monitor, y por lo tanto no se vea la barra inferior, existe la posibilidad de mostrar el icono en la barra superior del panel. Esta opción se activará añadiendo al parámetro **layout** del plugin **cwbarrasuppanel** el siguiente selector de css **show-modified**

```
{cwbarrasuppanel layout="show-modified" titulo="Panel listado"}
```

Figura D.6. Panel modificado, icono barra superior.



## D.5.2. Ubicación botonera: Nueva búsqueda, listado, edición.

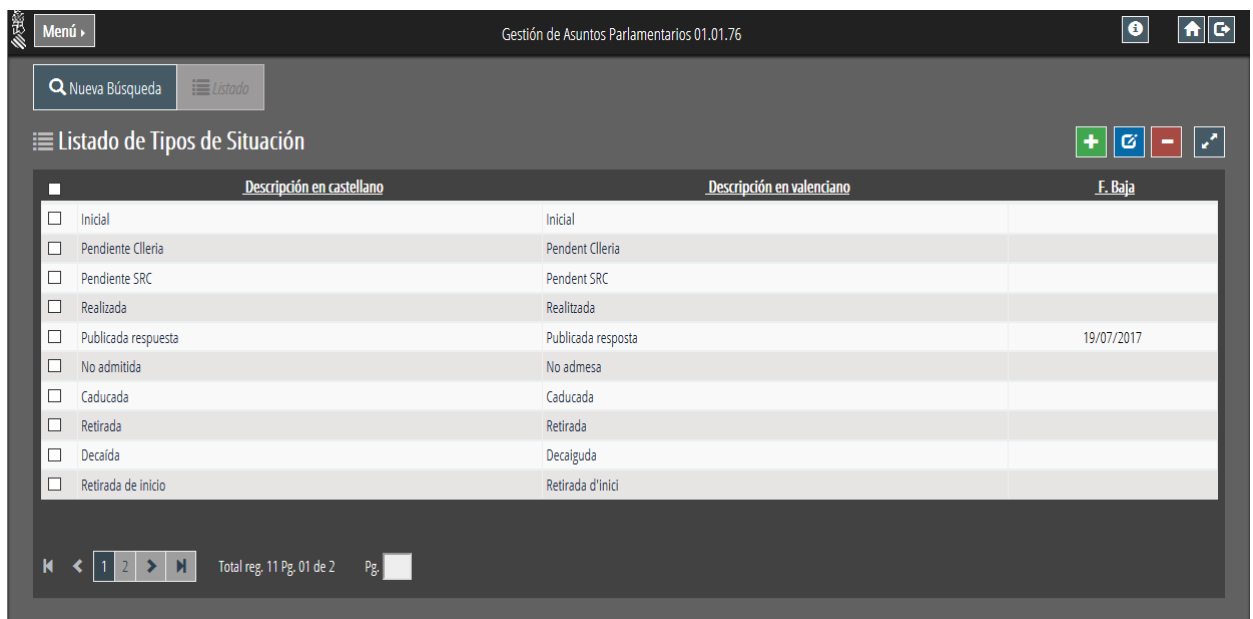
La botonera que aparece justo debajo de la barra superior de la ventana (Nueva búsqueda, listado, edición) puede visualizarse en formato maximizado y minimizado.

- **Formato maximizado botonera.**

Es la visualización clásica de gvHIDRA, por lo tanto, es la opción por defecto del framework. En este caso el parámetro **layout** deberá contener el valor **botonera-classic**

```
{cwventana layout="botonera-classic" titulo="APL"
 tipoAviso=$smtty_tipoAviso codAviso=$smtty_codError descBreve=
 $smtty_descBreve textoAviso=$smtty_textoAviso onLoad=$smtty_jsOnLoad}
```

**Figura D.7. Formato maximizado botonera (custom darkStyle).**



- **Formato minimizado botonera.**

En este caso la botonera aparecerá minimizada, es decir, los botones solamente contendrán los iconos que acompañan al texto en la versión anterior. Para conseguirlo, el parámetro **layout** deberá contener el valor **botonera-tabs**

```
{cwventana layout="botonera-tabs" titulo="APL"
 tipoAviso=$smtty_tipoAviso codAviso=$smtty_codError descBreve=
 $smtty_descBreve textoAviso=$smtty_textoAviso onLoad=$smtty_jsOnLoad}
```



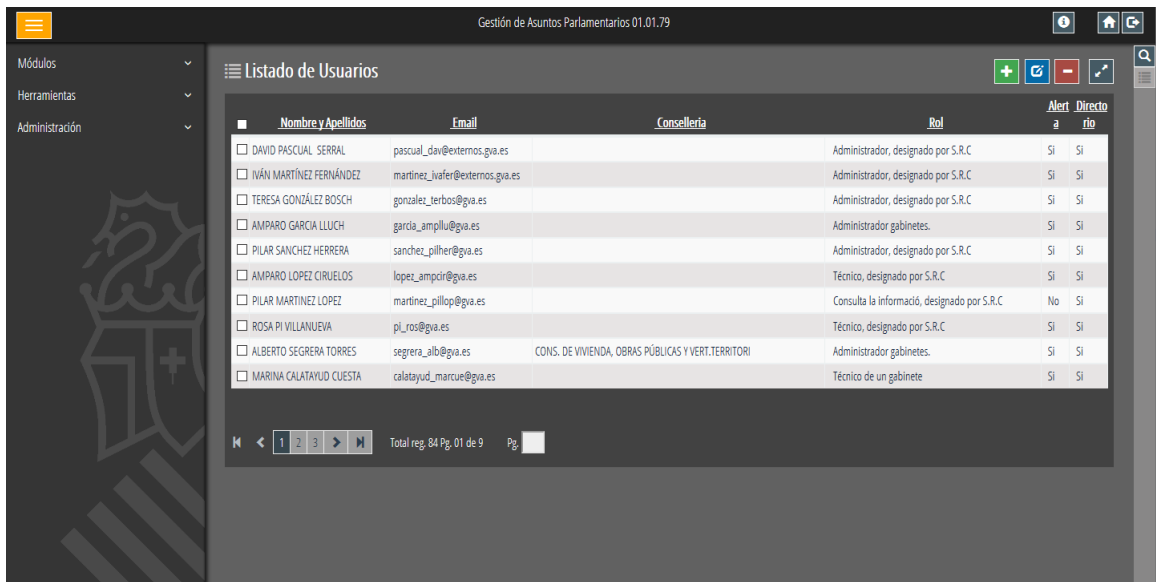
**Figura D.8. Formato minimizado botonera (custom darkStyle).**



Si se combina la botonera minimizada con el menú vertical, la botonera se colocará en el lado derecho de la pantalla de forma vertical.

```
{cwventana layout="vertical-menu botonera-tabs" titulo="APL"
 tipoAviso=$smtyp_tipoAviso codAviso=$smtyp_codError descBreve=
 $smtyp_descBreve textoAviso=$smtyp_textoAviso onLoad=$smtyp_jsOnLoad}
```

**Figura D.9. Formato minimizado botonera con menú vertical (custom darkStyle).**



### D.5.3. Redimensionar paneles.

En la barra superior de los paneles aparecerá, por defecto, un botón para maximizar/minimizar el panel.

**Figura D.10. Botón redimensionar.**



Si no se quiere que aparezca este botón en la barra superior únicamente se tendrá que añadir al parámetro *layout* el valor **no-maximize**.

```
{cwbarrasuppanel layout="no-maximize" titulo="Panel listado"}
```

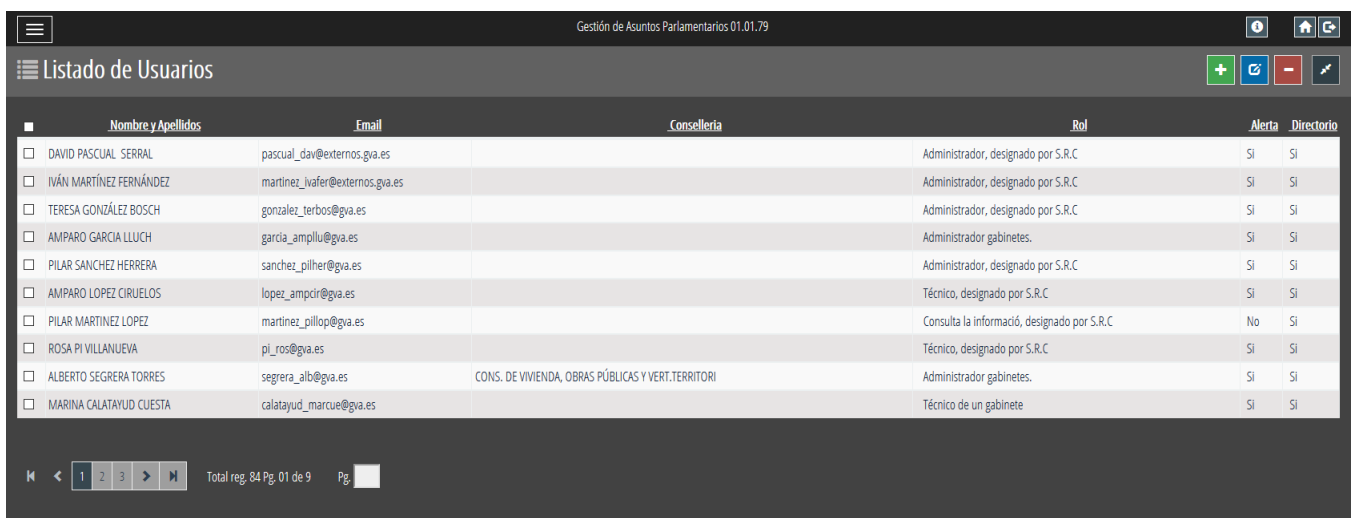
- **Panel minimizado**, estado por defecto.

**Figura D.11. Panel minimizado (estado por defecto).**



- **Panel maximizado**.

**Figura D.12. Panel maximizado.**



**D.5.4. Conteo de registros en las solapas de los detalles.**

Las pestañas de los detalles pueden ahora mostrar el n.º de tuplas que devolvería la consulta al pinchar en cada pestaña. Para mostrar el conteo en las pestañas, se ha añadido un parámetro booleano al método ‘addSlave’ que se invoca en la clase manejadora que actúa como maestra, cuyo valor por defecto es false y que puede cambiarse a true para activar el conteo de tuplas en dicha relación:

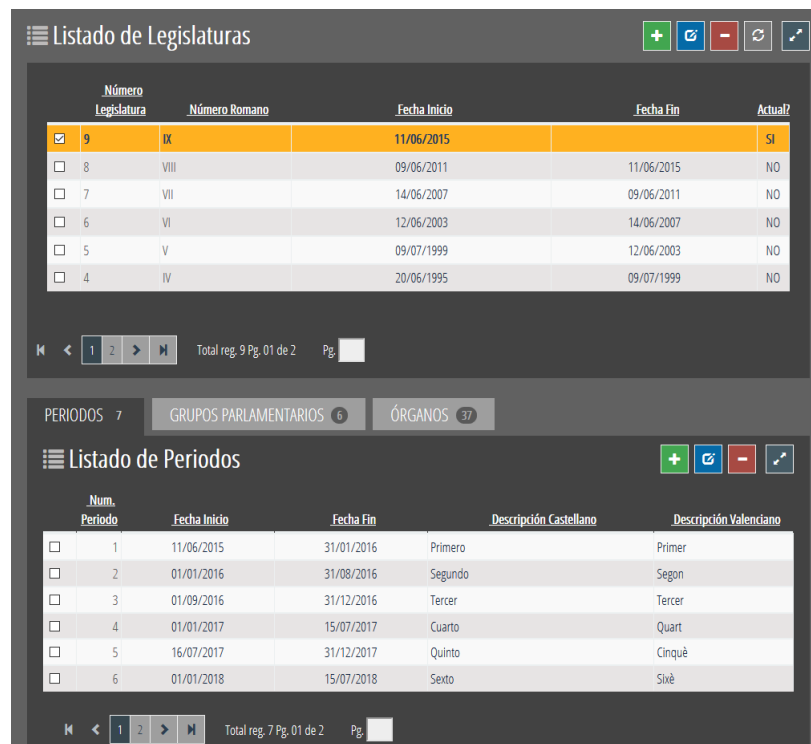
```
public function addSlave($nombreClaseManejadora, $listasCamposPadre,
 $listasCamposHijo, $contarTotalTuplas=false, $agruparConteo=true)
```

- **\$contarTotalTuplas:** Campo booleano opcional con el que se indicará si se quiere el conteo o no en las pestañas del detalle . Por defecto es false.

Para el conteo se construirá una consulta que por defecto utilizará las searchQueries() definidas en las distintas clases detalle/hija (definidas en el constructor de cada clase manejadora con los métodos 'setSelectForSearchQuery' y 'setWhereForSearchQuery'). Si quisiera optimizarse la consulta de conteo en cualquiera de los detalles, se puede utilizar un nuevo método 'setSelectForDetailCount', de funcionamiento similar a 'setSelectForSearchQuery' pero que se usará solo para el conteo de tuplas al actuar como detalle/hija. Si no se especifica nada en 'setSelectForDetailCount', se usará lo especificado en 'setSelectForSearchQuery'. En cuanto a las condiciones de filtro de la búsqueda (aka. sección sql WHERE), se utilizarán las mismas que para la consulta de búsqueda normal, es decir, se usarán las especificadas en 'setWhereForSearchQuery' y en los filtros. De ese modo, garantizaremos que se filtran siempre las mismas tuplas en el conteo que en la búsqueda.

- **\$agruparConteo:** Campo booleano opcional que agrupa la consulta de conteo de tuplas junto con la consulta de otros detalles. Por defecto es true.

Figura D.13. Maestro/nDetalles con conteo de registro en cada detalle.



## D.6. Configuración nuevas funcionalidades en paneles tabulares.

### D.6.1. Opciones de selección de registros en un tabular.

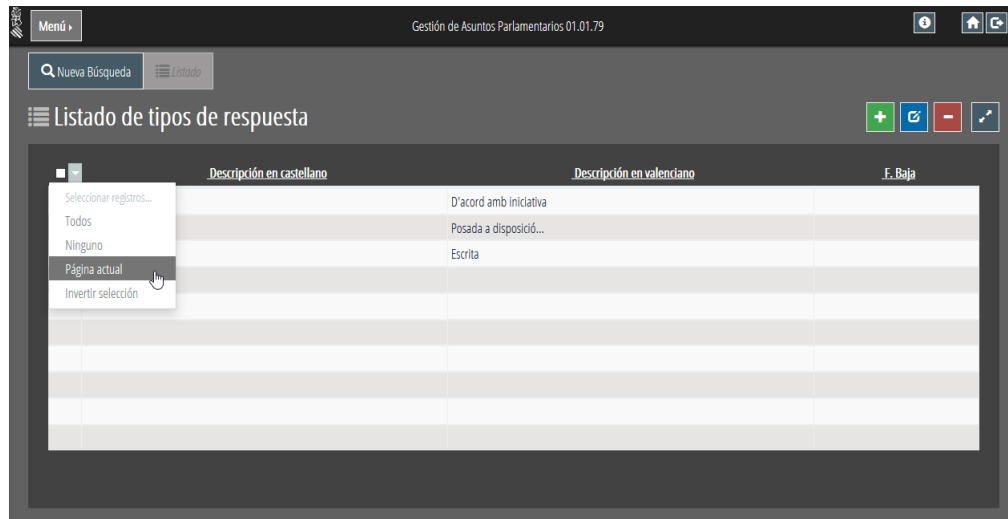
Cuando nos encontramos en un panel tabular, hasta ahora, se podía trabajar con más de un registro a la vez (parámetro *conCheckTodos=true* del plugin *cwtabla*) o en exclusividad con uno solo (parámetro *seleccionUnica=true* del plugin *cwtabla*).

En esta versión se amplían las posibilidades de selección de registros a los siguientes conjuntos: "Todos", "Ninguno", "Página actual", "Invertir selección". Para activar esta posibilidad en un panel tabular el valor del parámetro *con-*

*CheckTodos* debe tener el valor `multiSelect`. De este modo, junto al checkbox de selección de todas las tuplas de la tabla (`checkAll`) aparecerá un desplegable con opciones de selección múltiple sobre las tuplas

```
{cwtabla conCheckTodos="multiSelect" ... }
```

**Figura D.14. Panel tabular con opciones de selección de registros.**



## D.6.2. Selección directamente en la fila.

Tradicionalmente, en gvHIDRA, en un panel tabular para seleccionar un registro sobre el que trabajar (editar, eliminar) se selecciona dicho registro marcando el checkbox de la fila y pulsando el botón de la operación a realizar. En esta versión se ha incorporado la funcionalidad de poder realizar estos dos pasos en uno solo, haciendo click o doble click en cualquier parte de la fila. Para ello existen dos nuevos parámetros para el plugin `cwfila` que son `onRowClick` y `onRowDbClick`, respectivamente.

El valor de estos parámetros será el `id` del botón que queramos que se ejecute al hacer click sobre la fila.

- **Un solo click.**

Como se deduce de su nombre, con un solo click en la fila se realizará la acción que se haya indicado en el parámetro `onRowClick`.

```
...{cwbotontooltip id="btn_Modificar" iconCSS="glyphicon glyphicon-edit" titulo=#gvhlang_modificar# funcion="modificar" accion="modificar"
 actuaSobre="ficha" action="editar"}
{cwtabla ...}
 {cwfila tipoListado="false" onRowClick="btn_Modificar"}
 { * ... * }
 {/cwfila}
{/cwtabla}
```

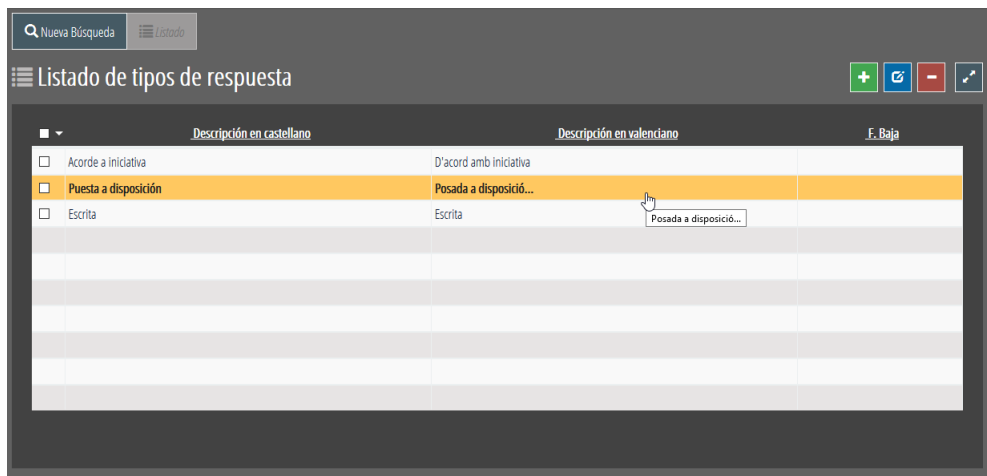
- **Doble click.**

Como se deduce de su nombre, con un doble click en la fila se realizará la acción que se haya indicado en el parámetro `onRowDbClick`.

```
...{cwbotontooltip id="btn_Modificar" iconCSS="glyphicon glyphicon-edit" titulo=#gvhlang_modificar# funcion="modificar" accion="modificar"
 actuaSobre="ficha" action="editar"}
{cwtabla ...}
 {cwfila tipoListado="false" onRowDbClick="btn_Modificar"}
 { * ... * }
```

```
{/cwfila}
{/cwtabla}
```

Figura D.15. Panel tabular con selección en fila activado.



### D.6.3. Mostrar número total de registros en un tabular.

En determinadas ocasiones puede interesar conocer el número total de registros aunque no se visualicen todos en una sola página. Para ello existe un nuevo parámetro para los plugins `cwtabla` y para `cwpaginador` llamado `conTotalRegistros` al que se le pueden asignar los siguientes valores:

- **always:** Mostrará siempre el número total de registros en el bloque al que afecte.
- **never:** No mostrará nunca el número total de registros en el bloque al que afecte.
- **only-multipage:** Sólo mostrará el número total cuando haya más de una página de datos.
- **only-singlepage:** Sólo mostrará el número total cuando haya una única página de datos.

El valor por defecto de `conTotalRegistros` para `cwtabla` es `never`, mientras que para `cwpaginador` es `only-multipage`

Figura D.16. Panel tabular con total de registros en `cwtabla`.

```
{cwtabla conCheck="true" conCheckTodos="true" numFilasPantalla="6" datos=
$smty_datosTabla conTotalRegistros="always" }
 {cwfila tipoListado="false" }
 { * ... * }
 {/cwfila}
 {cwpaginador iconCSS="true" conTotalRegistros="never" }
{/cwtabla}
```



**Figura D.17. Panel tabular con total de registros en cwpaginador.**

```
{cwtabla conCheck="true" conCheckTodos="true" numFilasPantalla="6" datos=$smtty_datosTabla}
 {cwfila tipoListado="false"}
 {* ... *}
 {/cwfila}
 {cwpaginador iconCSS="true" conTotalRegistros="only-multipage"}
{/cwtabla}
```



## D.6.4. Alineación de campos en un panel tabular.

En esta versión se incluyen unos estilos para que podamos alinear los campos por columnas en un panel tabular. Para utilizarlo se ha de incluir el estilo en el parámetro **class** con alguno de los siguientes valores: ['column-align-left', 'column-align-right', 'column-align-center']

**Figura D.18. Alineación campos en panel tabular.**

```
{cwcampotexto nombre="nombre" class="column-align-left" editable="true"
size="10" label="Nombre" dataType=$dataType_Tabular.nombre}
{cwcampotexto nombre="apel1" class="column-align-center" editable="true"
size="10" label="Apellido 1" dataType=$dataType_Tabular.apel1}
{cwcampotexto nombre="ape2" class="column-align-right" editable="true"
size="10" label="Apellido 2" dataType=$dataType_Tabular.ape2}
```



## D.7. Nuevas operaciones y parámetros en plugins.

### D.7.1. cwinfocontenedor. Operaciones getValue() y setValue().

Este plugin pasa de ser meramente informativo sobre el que no se podía actuar a poder tener un contenido dinámico. Desde la clase manejadora se podrá acceder a su contenido con los métodos **getValue()** y **setValue()**. El

único requisito es añadirle como segundo parámetro **"external"**, esto es porque se trata como un campo externo, no del conjunto de datos resultado de la consulta.

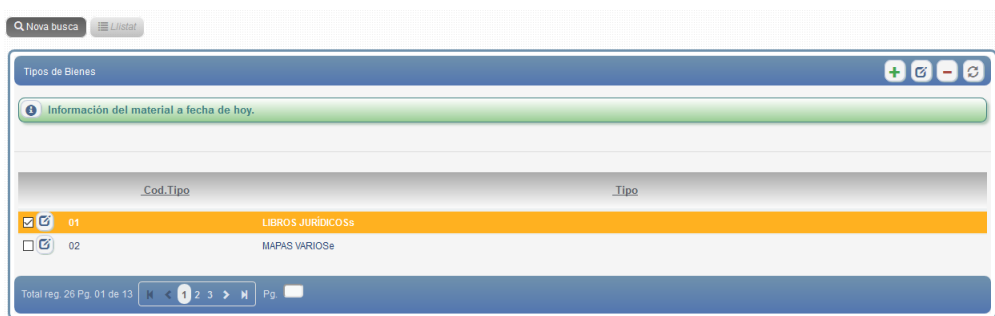
```
$contenedor = $objDatos->getValue ('infoContenedor1', 'external');

$objDatos->setValue (
 'infoContenedor2' ,
 'Se muestra la información correspondiente al día de hoy'
);
```

Por otra parte se añade el parámetro **"class"** para poder personalizar con css el plugin.

**Figura D.19. Plugin cwinfocontenedor.**

```
{cwinfocontenedor id="infoFicha" class="information" value="Información del material a fecha de hoy."}{/cwinfocontenedor}
```



## D.7.2. cwinformation. Operaciones `getValue()` y `setValue()`. Parámetros nuevos.

Este plugin pasa de ser meramente informativo sobre el que no se podía actuar a poder tener un contenido dinámico. Desde la clase manejadora se podrá acceder a su contenido con los métodos `getValue()` y `setValue()`, en este caso el valor sí puede pertenecer al conjunto de datos del resultado de la consulta.

El plugin también ofrece más posibilidades de diseño asociadas a los siguientes parámetros que se le han añadido:

- **posicion:** Establecerá la posición de la capa, donde se mostrará el contenido de "value", respecto al botón tooltip.

Los valores son: ["top", "right", "bottom", "left"]

- **size:** Valor para poder asignar tamaño a la columna en el caso de estar en un panel tabular.
- **trigger:** Momento en el que se mostrará el texto informativo.

Los valores son: ["focus", "hover"]

**Figura D.20. Plugin cwinformation.**

```
{cwinformation iconCSS="glyphicon glyphicon-info-sign" posicion="right"
nombre="mail" value=$defaultData_Tabular.mail
dataType=$dataType_Tabular.mail}
```



### D.7.3. cwbotontooltip con funcionamiento independiente del estado del panel.

Para permitir más versatilidad al plugin cwbotontooltip, cuando éste se encuentra dentro de un panel tabular o ficha, se le añade el parámetro **dependPanel**, este parámetro permitirá definir si el funcionamiento del botón es independiente o no del estado del panel. Es decir, si estará o no activo a pesar de que el panel no esté en modo de modificación o de inserción. Si *dependPanel* tiene el valor **"false"**, el botón estará **activo siempre**, es decir no dependerá de que el panel esté en modo inserción o edición.

Este parámetro también se añade para el plugin **cwcampotexto** pero solamente tendrá utilidad cuando el campo sea de tipo fecha, ya que el funcionamiento de si está activo o no recaerá sobre el botón tooltip del calendario que aparecerá al lado del campo.

```
{cwbotontooltip id="verInformation" dependPanel="false" iconCSS="glyphicon
glyphicon-refresh" accion="particular" action="verInformation"
actuaSobre="ficha" titulo="ver información"}
```

### D.7.4. Visibilidad de etiquetas que acompañan componentes.

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 es utilizar el parámetro y fijar su valor a false, así, evitaremos que se muestre el texto asociado.

Para ello se ha añadido un parámetro **"showLabel"** a todos los componentes que pueden aparecer dentro de un panel (cwareatexto, cwcampotexto, cwbotontooltip...), con él se podrá indicar si se quiere o no mostrar la etiqueta asociada al componente.

```
{cwcampotexto nombre="apel" editable="true" size="10" label="Apellido
1" showLabel="false" dataType=$dataType_Tabular.apel}
```

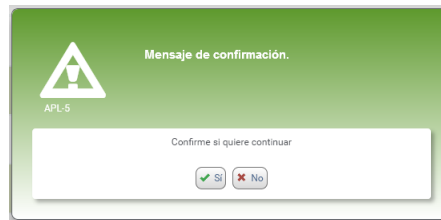
### D.7.5. Parámetro confirm en botones tooltip.

El parámetro **"confirm"** hasta ahora solo se podía utilizar en el plugin cwboton, pero a partir de esta versión se ha incluido en el plugin **cwbotontooltip**. A este parámetro se le debe pasar el código del mensaje que se quiere mostrar (p.ej "APL-5"), por lo tanto al pulsarlo saltará la ventana con el mensaje, será un mensaje de solicitud de confirmación para ejecutar o no la acción del panel.



**Figura D.21. Mensaje de confirmación.**

```
{cwbotontooltip iconCSS="glyphicon glyphicon-plus" texto="Particular"
class="button" accion="particular" action="MsgConfirm" confirm="APL-5" }
```



## D.8. Debug en desarrollo.

### D.8.1. Debug de plantillas en tiempo de ejecución en entorno de desarrollo.

Debug en tiempo de ejecución de plantillas Smarty en entorno local y de desarrollo (NO EN ENTORNO DE PRODUCCIÓN): Para poder ver, en tiempo de ejecución, la ventana de debug de un formulario Smarty basta con añadir el parámetro ‘SMARTY\_DEBUG’ a la ruta URL en la barra del navegador.

Por ejemplo, si la url de la aplicación fuera...

```
http://localhost/app/index.php?view=views/Vista.php
```

para mostrar la ventana de debug habría que añadirle ‘&SMARTY\_DEBUG’ :

```
http://localhost/app/index.php?view=views/Vista.php&SMARTY_DEBUG
```

Debido a que se muestran detalles internos de la aplicación, por seguridad este modo de depuración solo está disponible en los entornos local y de desarrollo, y por tanto NO ESTÁ DISPONIBLE EN ENTORNO DE PRODUCCIÓN. Esto es, solo está activado en los casos en que, en el fichero de configuración de la aplicación (‘gvHidraConfig.inc.xml’ o similar), la propiedad ‘p\_entorno’ tenga el valor ‘LOCALHOST’ o ‘DESARROLLO’:

```
<property id="p_entorno">LOCALHOST</property>
<property id="p_entorno">DESARROLLO</property>
```

### D.8.2. Debug de JavaScript.

Si el navegador soporta el debugger se puede utilizar *console.log()* para mostrar valores de JavaScript en la ventana de debugger. Cuando estamos desarrollando nos puede ser útil mostrar mensajes de este tipo. En esta versión para depurar mientras se desarrolla se pueden activar los mensajes de consola que por defecto tiene el framework con la directiva **logJSSettings** en el fichero de configuración de la aplicación **externo.gvHidraConfig.inc.xml**.

Activar el log:

```
<logJSSettings status='LOG_ALL'>
```

Desactivar el log:

```
<logJSSettings status='LOG_NONE'>
```

Si se quiere hacer uso en la consola JavaScript en el JavaScript que se incluya en alguna plantilla, es aconsejable utilizar la función `gvh.showConsoleMsg(type,msg)`. Ya que al hacer uso de este método funcionará el activar o no del log con la variable `logJSSettings`. Esta función necesita de dos parámetros:

- **type**: Será el tipo de mensaje que se mostrará en la consola.

Los valores posibles son los siguientes:

- **log**: Mensajes generales de información.
- **error**: Cuando queramos destacar que se ha producido un error, el mensaje en consola aparecerá en rojo.
- **warn**: Muestra un mensaje de advertencia.
- **dir**: Muestra un listado interactivo de las propiedades de un objeto JavaScript específico.
- **table**: Muestra datos tabulares en forma de una tabla.
- **beginGroup**: Crea un grupo en el que indenta todos los mensajes que se encuentren hasta que se indique cierre del grupo.
- **endGroup**: Finaliza el grupo abierto con "beginGroup".
- **msg**: Mensaje que queremos que se visualice en la consola.

Ejemplos:

Error:

```
gvh.showConsoleMsg('error',' ¡¡Falta la indicación del campo destino de la
ventana de selección!!');
```

Grupo:

```
gvh.showConsoleMsg('beginGroup',' ** gvH_panel.js - changeField() ');
...
gvh.showConsoleMsg('log', ' FIELD: '+idCampo);
...
gvh.showConsoleMsg('error', '¡¡¡ changeField() - '+campoJSON+'no existe este
campo en el registro '+regJSON+' !!!');
...
gvh.showConsoleMsg('endGroup');
```