RhinOS Developer Documentation

Josep Sanz Campderrós (www.saltos.net)

Índice

1. Framework RhinOS

El framework RhinOS, permite la creación de sites y aplicaciones online con las siguientes características:

- CMS (Content Manager System):
  - Parametrización: Permite la personalización de los listados y formularios.
  - Personalización: Permite aplicar cambios en la capa de negocio sin modificar el core, además de desarrollar aplicaciones a medida usando las API's del CMS.
  - Mantenimiento: Permite aplicar evolutivos con bajos costes de desarrollo.
  - Aplicaciones extras:
    - Configuración: Permite el control de los parámetros de configuración.
    - Usuarios: Permite la administración de usuarios y permisos.
    - Literales: Permite el control de los literales del portal
    - Fotos y documentos: Permite el control de contenidos multimedia
    - Google Maps: Permite la integración con servicios como Google Maps.
    - Mailings: Permite la integración de sistemas de mailings masivos.
- CPS (Content Publishing System):
  - Modular: Separación de la capa de presentación y capa de negocio
  - Personalización: Permite personalizar el portal y/o aplicación online para cada cliente usando lo que se desee en cada proyecto.
  - Mantenimiento: La arquitectura de desarrollo de RhinOS, permite un fácil y robusto mantenimiento y aplicación de evolutivos con bajos costes de desarrollo.
  - Aplicaciones extras: Permite crear módulos de programación que serán empleados desde los templates de forma fácil y rápida.

La documentación disponible del framework es la siguiente:

- Módulo AMDIN: Funcionamiento del CMS.
- Módulo SITE: Funcionamiento del CPS.
  - Módulo DBAPP2: Procesado de consultas al SGBD y generación de resultados.
  - Módulo CAPTCHA: Generación de imágenes CAPTCHA.
  - Módulo CONFIG: Acceso a variables de configuración.
  - Módulo EXTCACHE: Sistema de cache para acelerar los accesos a backoffices.

- Módulo FILES: Acceso a imágenes y ficheros.
- Módulo FILTER: Control de salida mediante condiciones.
- Módulo INTRANET: Control de zonas públicas y privadas mediante identificación.
- Módulo LABELS: Acceso a literales.
- Módulo QUERY: Automatización de consultas como INSERT, UPDATE y DELETE.
- Módulo SEND: Envío de correos con controles adicionales.
- Módulo SESSIONS: Acceso al sistema de sesiones (que se guarda en el SGBD).
- Módulo TIENDA: Procesado de peticiones y gestión de una tienda virtual.

2. Módulo ADMIN (CMS)

RhinOS, proporciona una herramienta de administración de contenidos tipo CMS, que permite las siguientes características:

- Personalización de la pantalla de aplicaciones.
- Personalización de los listados (campos, títulos, referencias a otras tablas, posiciones y dimensiones de las columnas, tipos de dato, ...)
- Personalización de los formularios de creación, consulta y edición (campos, titulos, referencias a otras tablas, posiciones, tipos de datos, campos únicos y campos obligatorios, ...)
- Aplicaciones a medida como:
  - Gestión de literales.
  - Gestión de fotos y documentos.
  - Gestión de mapas.
  - Gestión de parámetros de configuración.
  - Gestión de usuarios y roles.
- Sobrecarga de aplicaciones genéricas mediante el uso de las librerías del módulo.
- Creación de nuevas aplicaciones mediante el uso de las librerías del módulo.
- Sistema de mensajería entre los administradores y los usuarios.
- Sistema de bloqueo del aplicativo para tareas de mantenimiento.

- Descargas de copias de seguridad de toda la base de datos o de la configuración del módulo.
- Carga desde fichero y edición online de la configuración del módulo.

La estructura de tablas que usa este módulo es la siguiente:

- **Tablas db_*:** Todos los parámetros de configuración de este módulo se encuentran guardados en las tablas de la base de datos con el prefijo db_.
- **Tablas def_*:** Existen un conjunto de tablas con el prefijo def_, que permiten a la aplicación disponer de datos guardados que el usuario no puede (ni debe) modificar y que pertenecen al conjunto de datos de integridad de la aplicación como son los tipos de datos que dispone la aplicación, iconos disponibles para la aplicación, tamaños para los listados, tipos de datos dinámicos, roles y permisos de acceso.
- **Tablas tbl_*:** Estas son las tablas destinadas a aplicaciones del nivel de usuario.

La aplicación dispone de un sistema de auto-instalación que permite generar las tablas necesarias para su uso y cargar un conjunto de datos por defecto que hacen que la puesta en marcha del módulo sea automática. Esto permite al integrador crear y popular inicialmente las estructuras de datos que requerirá para su funcionamiento con sólo configurar la conexión a la base de datos.

Se pueden emplear otros prefijos para la creación de aplicaciones del nivel de usuario. Típicamente, se emplea el prefijo tbl_* para estas aplicaciones, pero se podrían emplear otros como app_*, tie_*, pro_* o lo que se desee en cada caso.

Notas de este módulo:

- Esta aplicación, generá una configuración inicial que a su vez, define como acceder a las tablas internas de configuración del módulo.
- En esta documentación se hará explicación del contenido que debe contener cada tabla de configuración (db_*), pero se pueden administrar estas tablas usando las aplicaciones que el módulo instala en el momento de la primera ejecución.
- Existen 4 maneras de alterar la configuración de este módulo:
  1. Usando las aplicaciones que instala inicialmente el módulo como son las aplicaciones de "Configuración", "Tablas", "Listados", "Formularios", "Relaciones" y "Dinamics".
  2. Editando la configuración a bajo nivel con la aplicación de "Editar configuración". Esta aplicación mostrará un fichero de texto con una linea por registro donde se podrán modificar, añadir o borrar lineas de configuración de cada tabla.

3. Subiendo un fichero de configuración usando la aplicación de "Upload configuración". Esta aplicación permitirá seleccionar un fichero del sistema local y subirlo al servidor para posteriormente cargar la configuración en la base de datos.
4. Generando un fichero de configuración en el directorio admin que se llame NOMBREBASEDEDATOS_db_spec.txt. Este fichero será cargado cada vez que se detecte un cambio en el contenido del mismo. La existencia de este fichero, dejará bloqueada las opciones 1, 2 y 3 de actualización de la configuración del módulo.

- Las opciones 2, 3 y 4 están protegidas contra errores en la nueva configuración. En caso de intentar cargar una configuración que genere errores, el sistema informará del error y restaurará la configuración existente en el momento del intento de cargar la nueva configuración.

2.1. Tipos de datos básicos de la aplicación

A continuación se listan los tipos básicos que RhinOS maneja:

- **text:**
  - Este tipo de dato define un campo de tipo texto en una línea
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''
- **textarea:**
  - Este tipo de dato define un campo de tipo texto en tipo multi-línea
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''
- **file:**
  - Este tipo de dato define un campo de tipo fichero
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''
  - Además, para hacer uso de este tipo de datos, es necesaria la existencia de 3 campos adicionales cuyo nombre sea el mismo pero con los sufijos:
    - _file (de tipo TEXT NOT NULL DEFAULT '')
    - _type (de tipo TEXT NOT NULL DEFAULT '')
    - _size (de tipo INTEGER NOT NULL DEFAULT '0')
- **photo:**
  - Este tipo de dato define un campo de tipo foto
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''.
  - Además, para hacer uso de este tipo de datos, es necesaria la existencia de 3 campos adicionales cuyo nombre sea el mismo pero con los sufijos
    - _file (de tipo TEXT NOT NULL DEFAULT '')

    - _type (de tipo TEXT NOT NULL DEFAULT '')
    - _size (de tipo INTEGER NOT NULL DEFAULT '0')

- **time:**
    - Este tipo de dato define un campo de tipo hora
    - El campo debe ser de tipo TIME NOT NULL DEFAULT '00:00:00'
- **date:**
    - Este tipo de dato define un campo de tipo fecha
    - El campo debe ser de tipo DATE NOT NULL DEFAULT '0000-00-00'
- **color:**
    - Este tipo de dato define un campo de tipo color
    - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''
    - Este campo permitirá contener un código hexadecimal como representación del color que guarda o una cadena vacía.
- **select:**
    - Este tipo de dato define un campo de tipo lista
    - El campo debe ser de tipo INTEGER NOT NULL DEFAULT '0'
    - Este tipo de dato debe tener una línea de configuración en la aplicación de relaciones (db_selects) que servirá para obtener el texto a mostrar y el identificador que se guardará en la base de datos como relación a otra tabla.
- **multiselect:**
    - Este tipo de dato es similar al anterior pero permite la relación con n datos de la tabla referenciada.
    - Este campo debe ser de tipo TEXT NOT NULL DEFAULT ''
    - Los valores en este campo se guardarán como una lista separada por comas con los identificadores de la relacion con la otra tabla
- **boolean:**
    - Este tipo de dato define un campo de tipo si/no (booleano)
    - El campo debe ser de tipo INTEGER NOT NULL DEFAULT '0'
    - Su representación en la aplicación sólo le permitirá usar valores como '0' para un valor falso y '1' para un valor cierto
- **password:**
    - Este tipo de dato define un campo de tipo contraseña

- El campo debe ser de tipo TEXT NOT NULL DEFAULT ''
- La representación de este campo será en dos cajas de tipo clave que servirán para validar que se ha entrado correctamente la contraseña usando dos repeticiones

- **md5password:**
  - Identico que el campo password, pero su valor se guardará en la base de datos usando un HASH calculado por la función MD5
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''

- **sha1password:**
  - Identico que el campo password, pero su valor se guardará en la base de datos usando un HASH calculado por la función SHA1
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''

- **integer:**
  - Este tipo de dato define un campo de tipo entero
  - El campo debe ser de tipo INTEGER NOT NULL DEFAULT '0'
  - Además, el interfaz aplicará una mascara sobre el valor entrado que no permitirá entrar valores diferentes al esperado en el caso del entero

- **real, decimal, float y double:**
  - Este tipo de dato define un campo de tipo real
  - El campo debe ser de tipo DECIMAL(9,2) NOT NULL DEFAULT '0.00'
  - Además, el interfaz aplicará una mascara sobre el valor entrado que no permitirá entrar valores diferentes al esperado en el caso del real

- **ajaxselect:**
  - Este tipo de dato es para mostrar información adicional, con lo que no deberá existir ningún campo que guarde ningún dato para este tipo de dato (solo será de lectura).
  - Este tipo de dato, sólo permite mostrar los datos de otras tablas pero únicamente es de tipo lectura, es decir, que el usuario no podrá interactuar con estos datos y sólo deberán usarse para mostrar información adicional de alguna relación existente en el formulario.

- **ajaxfilter:**
  - Este tipo de dato es idéntico al tipo de dato select y/o multiselect que será cargado usando llamadas AJAX.
  - El campo debe ser de tipo INTEGER NOT NULL DEFAULT '0' o TEXT NOT NULL DEFAULT '' dependiendo de la relación que se desee establecer

- Para hacer uso de este tipo de dato, deberá de especificarse un campo de referencia usando la siguiente notación en el nombre del campo, campo1:campo2 donde
- campo1 será el nombre del campo que se desea mostrar
- campo2 será el nombre del campo que se usará como filtro para obtener la lista de campo1 que cumpla la condición de campo2.
- Para más información acerca de este tipo de dato, véanse los ejemplos de tipos de datos cargados por AJAX.

- **ajaxdinamic:**
  - Este tipo de dato define un tipo de dato de tipo conjunto que puede contener datos de tipo nativos como text, textarea, file y/o photo.
  - El campo debe ser de tipo TEXT NOT NULL DEFAULT ''
  - También se deberán definir los campos adicionales con los sufijos siguientes:
    - _count (de tipo INTEGER NOT NULL DEFAULT '0')
    - _count_subtype (de tipo INTEGER NOT NULL DEFAULT '0')
      - Deberá existir un campo como el anterior para cada subtipo de dato que se desee emplear en dinamics.
      - Estos subtipos deberán existir en la tabla db_dinamics.
      - Donde subtype deberá ser el nombre del subtipo de dato usado en dinamics como (esta lista es la lista por defecto de db_dinamics):
      - titulo (de tipo text)
      - subtitulo (de tipo text)
      - descripcion (de tipo textarea)
      - foto (de tipo photo)
      - fichero (de tipo file)
      - url (de tipo text)
      - video (de tipo text)
    - _count_data_0 (de tipo TEXT NOT NULL DEFAULT '')
    - _count_data_0_type (de tipo TEXT NOT NULL DEFAULT '')
    - _count_data_0_file (de tipo TEXT NOT NULL DEFAULT '')
    - _count_data_0_size (de tipo INTEGER NOT NULL DEFAULT '0')
    - Se deberán crear 4 campos de los tipos anteriores para cada posible dato dinamics que se desee guardar, teniendo que iterar en la creación de campos de la base de datos hasta obtener lo siguiente:
      - _count_data_(n-1) (de tipo TEXT NOT NULL DEFAULT '')
      - _count_data_(n-1)_type (de tipo TEXT NOT NULL DEFAULT '')
      - _count_data_(n-1)_file (de tipo TEXT NOT NULL DEFAULT '')

- ◇ _count_data_(n-1)_size (de tipo INTEGER NOT NULL DEFAULT '0')
- ◇ n será el valor máximo permitido de datos dinamics que se podrán guardar en el registro.

2.2. Aplicación de configuración (db_config)

Esta aplicación permite al módulo ADMIN definir variables que se convertirán en globales y que indicarán al CMS, por ejemplo, que imagen de FAVICON debe emplearse. Un ejemplo de su contenido puede ser:

```
#
TABLE db_config
#
SPEC "param","value"
ROW "pagename","BaseWEB"
ROW "pagemail","info@saltos.net"
ROW "admuser","admin"
ROW "pagestyle","redmond"
ROW "pagelimit","15"
ROW "floatmenu","0"
```

Como se puede apreciar, la columna param contiene el nombre de la variable y la columna value, contiene el valor de dicha variable.

2.3. Aplicación de tablas (db_tables)

Esta aplicación permite definir las aplicaciones que el usuario tendrá acceso (más adelante se explicará como asignar roles de acceso a los usuarios). Esta aplicación únicamente define el conjunto de aplicaciones disponibles que luego serán empleadas para acceder a las aplicaciones si el usuario dispone del suficiente número de privilegios.

Un ejemplo típico del contenido de esta aplicación es:

```
#
TABLE db_tables
#
SPEC "tbl","name","description","position","icon"
ROW "password.php","Cambiar contraseña","Para cambiar su clave de acceso","100","password.png"
ROW "contact.php","Soporte técnico","Enviar un e-mail al administrador<br/>del sistema","101","xfmail.png"
ROW "about.php","Acerca de RhinOS","Conozca los creditos de RhinOS<br>y los componentes que usa","102","khelpcenter.png"
ROW "edit_db_spec.adm","Editar configuración","Editar la configuración del admin","103","easymoblog.png"
ROW "swap_block_admin.adm","Bloquear Admin","Bloquear y desbloquear el admin","10 ","daemons.png"
ROW "upload_db_spec.adm","Upload configuración","Cargar configuracion del admin","105","download.png"
ROW "download_db_spec.adm","Download configuración","Descargar configuración del admin","106","3floppy_unmount.png"
ROW "download_backup.adm","Download backup","Descargar backup de la base de datos","107","3floppy_unmount.png"
ROW "catalog.php","Catalogo iconos","Catalogo con todos los iconos","108","icons.png"
ROW "db_config","Configuración","Configuración de parametros","200","file-manager.png"
ROW "db_users","Usuarios","Alta y baja de usuarios del admin","201","password.png"
ROW "db_perms","Permisos","Control de permisos del admin","202","password.png"
ROW "db_tables","Tablas","Aplicaciones visibles","203","desktop.png"
ROW "db_lists","Listados","Gestión de listados","20 ","desktop.png"
ROW "db_forms","Formularios","Administración de formularios","205","desktop.png"
ROW "db_selects","Relaciones","Relaciones de la base de datos","206","ksirtet.png"
ROW "db_notes","Notas","Notas a los usuarios del admin","207","knotes.png"
ROW "db_sessions","Sesiones","Tabla de sesiones","208","gpg.png"
ROW "db_dinamics","Dinamics","Tipos de datos dinámicos","209","kword.png"
ROW "tbl_config.php","Configuración","Gestione la configuración de su sitio web","300","advancedsettings.png"
```

```
ROW "tbl_users.php","Usuarios","Gestione los controles de acceso a esta aplicación","301","password.png"
ROW "tbl_google.php","Google Maps","Configuración la aplicación de Google Maps","302","access.png"
ROW "tbl_labels.php","Literales","Configure los textos de su portal desde aquí","303","ktouch.png"
ROW "tbl_files.php","Fotos y documentos","Gestión de fotos y documentos de su portal","30 ","camera_unmount.png"
ROW "tbl_dinamics","Páginas dinámicas","Gestione con total dinamismo su portal","305","kword.png"
ROW "tbl_mailing.php","Envio de mailings","Genera un nuevo mailing masivo"," 00","xfmail.png"
ROW "tbl_queue","Cola de envios","Contiene los emails a enviar y enviados"," 01","xfmail.png"
ROW "tbl_response","Cola de respuestas","Esta cola contiene las respuestas de los envios"," 02","korn.png"
```

Esta aplicación define las posibles aplicaciones que podrá ejecutar el módulo. También proporciona la información necesaria para poder representar la pantalla de inicio del CMS donde se representarán todas las aplicaciones disponibles para el usuario con un nombre, una descripción y un icono.

La posición se determinará usando la columna position y permite generar grupos de aplicaciones. Para hacer uso de esta prestación, es necesario indicar la posición usando unidades de centena para determinar el grupo y el resto para determinar la posición dentro de su grupo.

Como se puede observar en el ejemplo anterior, esta configuración indica 4 grupos de aplicaciones que serán mostradas de forma gráfica como agrupaciones de aplicaciones.

Notas de esta aplicación:

- El valor de la columna icon debe contener el nombre del icono a mostrar en la aplicación con la extensión y debe existir en el directorio lib/crystal/SZxSZ donde SZ será 16, 32 y 48. En caso de no existir, se empleará el valor kblackbox.png (icono por defecto).
- También se permite el uso de la palabra clave "random" que indicará al admin que busque un icono de forma pseudoaleatoria para esa aplicación.

2.4. Aplicación de listados (db_lists)

Esta aplicación permite indicar para cada aplicación que configuración de listado se desea emplear. Un ejemplo de configuración de listado para una aplicación podría ser:

```
#
TABLE db_lists
#
SPEC "tbl","row","name","type","edit","size","position"
ROW "tbl_queue","subject","Subject","text","0"," 0%","1"
ROW "tbl_queue","from","From","text","0","30%","2"
ROW "tbl_queue","status","Enviado","boolean","0","10%","3"
ROW "tbl_queue","hora","Hora","time","0","10%"," "
ROW "tbl_queue","fecha","Fecha","date","0","10%","5"
```

En esta aplicación, deben emplearse el nombre de la tabla y el nombre de cada campo que se desee mostrar en el listado. Para cada combinación anterior, además, deben proporcionarse la información para mostrar el nombre del campo, el tipo, la posición y tamaño y si son capos editables.

Los tipos de datos que un listado puede emplear son los siguientes:

- **select**: La representación que se hará de este tipo de datos sera el valor de la relación a la que apunta (en caso de existir y contener valor). También se mostrará un icono que permitirá aplicar un filtro por el valor de este campo.
- **multiselect**: La representación que se hará de este tipo de datos sera el valor de la relación a la que apunta (en caso de existir y contener valor). En caso de que la relación apunte a varios registros de la tabla destino, se concatenarán los valores separados por comas (,).
- **file**: Mostrará el nombre del fichero y a su vez, se proporcionará un enlace para poder acceder al contenido del fichero para su descarga. El tooltip que se mostrará contendrá toda la información referente al fichero.
- **photo**: Mostrará el nombre con un pequeño preview de la imagen y a su vez, se proporcionará un enlace para poder acceder al contenido del fichero para su descarga. El tooltip que se mostrará contendrá toda la información referente a la imagen.
- **boolean**: Mostrará un checkbox con el valor que tiene el campo. También se mostrará un icono que permitirá aplicar un filtro por el valor de este campo.
- **ajaxboolean**: Idem que el anterior, con la peculiaridad de que será posible cambiar el valor de este booleano haciendo clic en el propio checkbox sin necesidad de hacer ninguna acción adicional. Es útil para permitir un rápido acceso a la opción de cambiar ese valor única y exclusivamente.
- **date**: Muestra la fecha en formato DD/MM/AAAA. También se mostrará un icono que permitirá aplicar un filtro por el valor de este campo.
- **time**: Muestra la hora en formato HH:MM (los segundos se omiten en este tipo de dato).
- **color**: Muestra el valor hexadecimal del color con un preview del mismo (sólo en caso de contener un valor diferente de vacio).
- **text / textarea**: Muestra el texto (y aplica técnicas de limpiado de entidades html para el caso del textarea). En todos los casos, realiza un cortado de la cadena que se mostrará si supera la longitud el tamaño especificado.
- **integer**: Muestra el valor del entero guardado.
- **real / decimal / float / double**: Muestra el valor del real guardado.

Adicionalmente, si la tabla que se desea configurar contiene un campo llamado _needed, se bloqueará la eliminación del campo si el valor del campo _needed es '1' y en caso de que sea '0', se permitirá un comportamiento normal con opción al borrado del registro. Esta prestación es útil cuando se crean registros en el momento del desarrollo y se desea evitar el el usuario pueda borrar ese registro (típicamente por que la programación buscará ese registro con ID conocido).

2.5. Aplicación de formularios (db_forms)

Esta aplicación permite indicar para cada aplicación que configuración de formulario se desea emplear. Un ejemplo de configuración de formulario para una aplicación podría ser:

```
#
TABLE db_forms
#
SPEC "tbl","row","name","type","needed","unique","position","noedit"
ROW "tbl_queue","hora","Hora","time","0","0","1","0"
ROW "tbl_queue","fecha","Fecha","date","0","0","2","0"
ROW "tbl_queue","host","Host","text","1","0","2","0"
ROW "tbl_queue","user","User","text","1","0","2","0"
ROW "tbl_queue","pass","Pass","text","1","0","2","0"
ROW "tbl_queue","from","From","text","1","0","3","0"
ROW "tbl_queue","fromname","FromName","text","1","0"," ","0"
ROW "tbl_queue","to","To","textarea","1","0","5","0"
ROW "tbl_queue","subject","Subject","text","1","0","6","0"
ROW "tbl_queue","fichero","Fichero","file","0","0","7","0"
ROW "tbl_queue","body","Body","textarea","0","0","8","0"
ROW "tbl_queue","status","Enviado","boolean","0","0","9","0"
```

En esta aplicación, deben emplearse el nombre de la tabla y el nombre de cada campo que se desee mostrar en el listado. Para cada combinación anterior, además, deben proporcionarse la información para mostrar el nombre del campo, el tipo, la posición, si son campos necesarios y/o unicos y si es un campo bloqueado de sólo lectura.

Los tipos de datos que un formulario puede emplear son los siguientes:

- **text**: Mostrará un campo de edición de una linea
- **textarea**: Mostrará un editor de texto avanzado (CKEditor)
- **select**: Mostrará una lista desplegable con los valores de la tabla referenciada. Si esta lista supera los 100 elementos, automáticamente se mostrará una caja de texto que permitirá filtrar los resultados que se mostrarán en la lista desplegable.
- **multiselect**: Mostrará dos cajas de listas con botones para permitir mover elementos entre las dos listas. Si esta lista de datos origen supera los 100 elementos, automáticamente se mostrará una caja de texto que permitirá filtrar los resultados que se mostrarán en la lista origen.
- **file**: Mostrará en caso de que exista valor, el fichero con sus datos adicionales, además de proporcionar un enlace de acceso al fichero para su descarga. También permitirá cambiar el fichero por uno nuevo que se seleccione del sistema local del cliente.
- **photo**: Mostrará en caso de que exista valor, la foto con sus datos adicionales, además de proporcionar un enlace de acceso a la foto para su descarga. También permitirá cambiar la foto por una nueva que se seleccione del sistema local del cliente.

- **password**: Mostrará 2 cajas de texto que permitirán establecer la contraseña en el campo especificado. Se solicitarán 2 valores para comprobar que sean idénticos antes de realizar la creación o modificación del valor en la base de datos.
- **md5password**: Idem que el tipo password, pero con la diferencia de que el valor será guardado aplicando la función de HASH en formato MD5.
- **sha1password**: Idem que el tipo password, pero con la diferencia de que el valor será guardado aplicando la función de HASH en formato SHA1.
- **boolean**: Mostrará 2 group buttons que permitirán establecer el valor Si (1) o No (0).
- **date**: Mostrará una caja de texto de una linea con el valor de fecha que exista en la DB. También mostrará un botón que abrirá un selector de fechas (datepicker de JQuery). También se aplicará un filtro para la escritura que forzará entrar un valor correcto.
- **time**: Mostrará una caja de texto de una linea con el valor de hora que exista en la DB. El valor empleará el formato HH:MM omitiéndose los segundos. Se mostrará adicionalmente un icono con un tooltip que indicará el formato para la fecha. También se aplicará un filtro para la escritura que forzará entrar un valor correcto.
- **color**: Se mostrará una caja de texto de una linea que permitirá entrar y/o mostrar el color existente en la DB. Se mostrará al lado de la caja de texto, otra caja con el preview del color y un icono que abrirá un selector de color (colorpicker de JQuery). También se aplicará un filtro para la escritura que forzará entrar un valor correcto.
- **integer**: Se mostrará una caja de texto de una linea que permitirá entrar y/o mostrar el valor del entero existente en la DB. También se aplicará un filtro para la escritura que forzará entrar un valor correcto.
- **real / decimal / float / double**: Se mostrará una caja de texto de una linea que permitirá entrar y/o mostrar el valor del real existente en la DB. También se aplicará un filtro para la escritura que forzará entrar un valor correcto.
- **ajaxselect**: Este tipo de dato se empleará cuando exista otro campo que es de tipo select y se desee mostrar información adicional de la relación que el otro campo crea al hacer una selección de valor referenciando a otra tabla. Puede emplearse si el valor que se muestra en el tipo select no es suficiente para determinar que valor se esta seleccionando o por si se desea mostrar información adicional de la relación establecida con el select. Para emplear este tipo de dato, no debe existir ningún campo que guarde el valor del mismo pues es de sólo lectura y se empleará para ampliar la información de la referencia.

- **ajaxfilter**: Este tipo de dato, se emplea cuando se desea emplear un dato de tipo select o multiselect y se desean que los datos de estos tipos de datos se filtren por otro dato existente en el formulario. Un ejemplo típico podria ser el mostrar un select con las provincias de España y emplear un ajaxfilter para mostrar un select con las poblaciones de la provincia seleccionada y a su vez, otro select con los códigos postales de la población seleccionada.

- **ajaxdinamic**: Este tipo de dato permite indicar que un campo contendrá una estructura de datos generada por el sistema dinamics permitiendo al usuario definir el orden y tipo de datos que deseen guardar. Para hacer uso de este tipo de dato, deben de existir los campos necesarios para el buen funcionamiento de dinamics.

2.6. Aplicación de relaciones (db_selects)

Esta aplicación permite definir las relaciones entre tablas, de forma que se consiguen 2 objetivos:

- Indicar al módulo ADMIN que relaciones existen entre las tablas para evitar borrar registros que puedan comprometer la integridad referencial del sistema.

- Indicar al módulo ADMIN que texto se debe mostrar y que valor se debe guardar y emplear para las relaciones definidas con tipos de datos de tipo select y multiselect.

El contenido que el módulo crea en esta aplicación en la instalación inicial es:

```
#
TABLE db_selects
#
SPEC "tbl","row","table_ref","value_ref","text_ref"
ROW "db_dinamics","type","def_dinamics","value","value"
ROW "db_forms","type","def_types","value","value"
ROW "db_forms","tbl","db_tables","tbl","tbl"
ROW "db_lists","size","def_sizes","value","value"
ROW "db_lists","type","def_types","value","value"
ROW "db_lists","tbl","db_tables","tbl","tbl"
ROW "db_notes","user","db_users","login","login"
ROW "db_perms","allow","def_allows","value","value"
ROW "db_perms","user","db_users","login","login"
ROW "db_perms","role","def_roles","value","value"
ROW "db_selects","tbl","db_tables","tbl","tbl"
ROW "db_selects","table_ref","db_tables","tbl","tbl"
ROW "db_tables","icon","def_icons","icon","icon"
ROW "tbl_config","deltag","tbl_config","param","param"
ROW "tbl_files","deltag","tbl_files","tag","tag"
ROW "tbl_labels","deltag","tbl_labels","tag","tag"
ROW "tbl_labels","deltag:delsec","tbl_labels","sec","multiselect"
ROW "tbl_response","id_queue","tbl_queue","id","subject"
```

2.7. Configuración del módulo

Este módulo debe configurarse usando el fichero config.php que existe en el directorio admin. A continuación se muestra el contenido del mismo:

```
<?php
// DATABASE
$_CONFIG["db"]["type"]="pdo_sqlite";
$_CONFIG["db"]["host"]="localhost";
$_CONFIG["db"]["user"]="baseweb";
$_CONFIG["db"]["pass"]="baseweb";
$_CONFIG["db"]["name"]="baseweb";
$admindir=str_replace(DIRECTORY_SEPARATOR,"/",getcwd());
$admindir=str_replace("/admin","",$admindir)."/admin";
$_CONFIG["db"]["file"]="$admindir/files/baseweb.db";
$_CONFIG["db"]["link"]=null;
// DEFINES
//define("DEMO",1);
//define("DEBUG",1);
// OVERLOADS
//define("__FORCE_ISMAIL__",true);
//define("__CHECK_REFERER__",0);
//define("__CHECK_DEVEL__",1);
//define("__PHPTHUMB_USM__",0);
// LANGS
//$_LANG=array("lang"=>"es");
//$_LANG=array("lang"=>"ca");
//$_LANG=array("lang"=>"en");
// AUTOMATE THE DEBUG FEATURE
if(!defined("DEBUG") && file_exists($admindir."/DEBUG")) {
define("DEBUG",file_get_contents($admindir."/DEBUG"));
}
?>
```

3. Módulo SITE

El módulo SITE, también llamado CPS o Content Publishing System, es un servidor virtual que permite atender a las peticiones http interpretando el tipo de contenido solicitado, enviando las cabeceras y generando los contenidos en caso necesario.

Para ello, se requiere de las siguientes características en el servidor web:

- Servidor Apache.
  - Activación de la directiva MultiViews: Esta directiva permite a RhinOS actuar como pasarela de todas las peticiones que se realicen para obtener contenidos (textos, imagenes, ficheros, videos, ...).
  - Activación de la directiva FollowSymLinks: Permitirá a RhinOS segmentar el código en una parte destinada al CORE y a otra para las personalizaciones de cada proyecto.

3.1. La directiva MultiViews

Esta directiva es sumamente importante para RhinOS pues le permite que un acceso a una url como:

```
http://localhost/baseweb/portal/inicio.htm
```

Donde en realidad solo exista:

```
http://localhost/baseweb/portal.php
```

Ejecute el código de portal.php y se defina una variable de entorno llamada PATH_INFO con el valor inicio.htm.

Esto es el principio básico de RhinOS, el cual intercepta cualquier petición que se realiza para generar las cabeceras y el contenido apropiado para cada momento.

De esta forma, un acceso a:

```
http://localhost/baseweb/portal/img/logo.png
```

Lo que hace en realidad es ejecutar el script portal.php definiendo la variable PATH_INFO con el valor img/logo.png y será el script ejecutado el encargado de servir el contenido esperado con las cabeceras del tipo de contenido esperado.

Además, RhinOS proporciona un sistema de generación de contenidos basado en un parser de pseudo-código que permite usar cache para acelerar su ejecución.

3.2. Variables de RhinOS

Variables de configuración RhinOS:

RhinOS, define si no se han definido las siguientes variables que afectarán en su comportamiento:

- Variables de acceso al SGBD:
  - `$_CONFIG["db"]["type"]="pdo_sqlite";`
  - `$_CONFIG["db"]["host"]="localhost";`
  - `$_CONFIG["db"][üser"]="baseweb";`
  - `$_CONFIG["db"]["pass"]="baseweb";`
  - `$_CONFIG["db"]["name"]="baseweb";`
  - `$_CONFIG["db"]["file"]=.ªdmin/database/baseweb.db";`
  - `$_CONFIG["db"]["link"]=null;`
- Defines y control de errores
  - `define("DEMO",1);`
  - `define("DEBUG",1);`
  - `error_reporting(E_ALL);`
- Variables de configuración:
  - `$timezone=.Europe/Madrid";`
  - `$plantillas="templates";`
  - `$inicio="inicio.htm";`
  - `$secundaria="secundaria.htm";`

Variables del RunTime de RhinOS:

Además de las variables de configuración que indicarán a RhinOS donde y como acceder a los diferentes recursos del sistema, tales como el SGBD o ficheros, también se definen variables que hacen a RhinOS poder acceder a otros valores necesarios para la evaluación de las peticiones entrantes:

- `$argv`: Estas variables contendrán el array de argumentos obtenidos de la url (en realidad, de la variable PATH_INFO que tan importante es para RhinOS). En adelante, se empleará el uso de ARGV[x] para referirse a $argv[x].
- `$_SESSION`: Este array contendrá las variables definidas en la sesión (equivalente al $_SESSION que por defecto define PHP pero en RhinOS, las sesiones se guardan en el SGBD usando a la implementación del handler de sesiones que proporciona RhinOS).
- `$_GET y $_POST`: Este array contiene las variables procedentes de los arrays que proporciona PHP por defecto.

La variable ARGV, contiene el array que se construye usando el PATH_INFO recibido en la petición. Este array se usa para conocer que tipo de contenido se esta solicitando y donde hay que hacerlo.

3.3. Estructura de las peticiones http

Como se ha explicado anteriormente, la directiva MultiViews permite a RhinOS conocer que petición se esta realizando. Esta petición es separada en argumentos que se encuentran definidos en el array ARGV.

RhinOS, define un modelo de servidor basado en la siguiente estructura de peticiones (ya que con ARGV[0] se puede determinar que tipo de contenido se esta solicitando):

- **img**: Se está solicitando un contenido de tipo imagen. Si el valor de ARGV[2] esta definido, se empleará la librería phpthumb para generar la salida.
- **img / ico / swf / flv**: Se envía el contenido del fichero solicitado .
- **rss / xml / xsl**: Se envia el resultado del parser del fichero solicitado.
- **htm**: Se envia el resultado del parser del fichero solicitado.
- **print**: Se envia el resultado del parser del fichero solicitado.
- **open / down**: Se envia el resultado del fichero solicitado.
- **js / json**: Se envia el resultado del parser del fichero solicitado.
- **css**: Se envia el resultado del fichero solicitado
- **cache**: Se envia el resultado de la minimización y de la cache del fichero solicitado

- El resto de peticiones, deben ser sobre un fichero que exista en el directorio de templates.

A continuación, se explicará con detalle que se hace con cada tipo de petición.

3.4. Peticiones a img/

La forma de realizar peticiones de imágenes que deseamos que sean procesadas por la librería gráfica, es la siguiente:

```
http://localhost/baseweb/portal/img/IMAGEN/ALTOxANCHO/OPCIONES/IMAGEN.FORMATO
```

Como se puede observar, el sistema generará un ARGV igual a:

- img
- IMAGE
- ALTOxANCHO
- OPCIONES
- IMAGEN.FORMATO

Donde:

- **IMAGEN** indica el nombre de fichero que debe existir en:
  - admin/files/IMAGEN
  - templates/img/IMAGEN
- **ALTOxANCHO** indica el tamaño de la imagen generada (original si no se desea cambiar la dimensión).
- **OPCIONES** son los parámetros adicionales que serán empleados para generar la imagen. Generalmente son efectos que permiten hacer los bordes redondeados, marcas de agua, ...
- **IMAGEN.FORMATO** es empleado para determinar que formato de salida se desea. Para ello, no importa el valor de IMAGEN pero sí el valor de FORMATO (que puede ser png, jpg, tiff, bmp, gif, aunque esto depende de las librerías gráficas instaladas en el servidor).

Si ARGV[2] esta definido, entonces se espera que este contenga el tamaño de la imagen que se va a generar. Si se desea que la librería gráfica no cambie las dimensiones de la imagen, se puede usar la palabra clave .ºriginal", la cual fuerza a que se apliquen el resto de efectos sobre la imagen sin cambiar las dimensiones de la misma.

Si no se emplea la palabra clave .ºriginal", ARGV[2] debe contener una cadena del tipo {ancho}x{alto}. Estos valores son comprobados para evitar ataques de tipo

DoS de forma que si el ancho o el alto es superior a 2000, entonces el sistema generará un error 404. En caso de especificar valores negativos para el ancho o el alto, también se generará un error 404.

Algunas prestaciones que todas las imágenes emplearán serán:

- Las imágenes que se generarán, emplearán un factor q=100 (máxima calidad).
- Se aplicará un filtro para realzar el contraste de la imagen (usm|80|0.5|3).

La opciones que permite la llamada a img/ cuando se indica un parámetro en ARGV[2] son:

- **ric/n**: redondeo de los bordes de la imagen de radio n. Es útil para obtener imagenes con los bordes redondeados.
- **over/imagen**: aplica el efecto de overlay usando el argumento adicional imagen. Es útil cuando se desea obtener una imagen cuyo resultado sea más elaborado que un simple efecto como RIC. El argumento adicional imagen, debe indicar el nombre de un fichero que debe encontrarse en el directorio img del template que lo use.
- **far**: rellena con color de fondo hasta obtener una imagen del tamaño ANCHO y ALTO. Este efecto mantiene la relación de aspecto de la imagen.
- **iar**: estira la imagen hasta obtener una imagen del tamaño ANCHO y ALTO. Este efecto altera la relación de aspecto de la imagen pudiendo generar imágenes con apariencia deformada.
- **backgroundcolor / background / bgcolor / bg**: especifica el color de fondo que se usará en los efectos. Debe ser un color en hexadecimal sin que le preceda el simbolo #.
- **foregroundcolor / foreground / fgcolor / fg**: especifica el color de primer plano que se usará en los efectos. Debe ser un color en hexadecimal sin que le preceda el simbolo #.
- **font / fn**: establece el nombre de la fuente a usar en la generación de textos en las imagenes. Debe ser un fichero .ttf y no se debe especificar la extensión, debe existir el fichero en el directorio fonts del template que lo use. El valor por defecto es DejaVuSans.
- **size / sz**: establece el tamaño de la fuente en la generación de textos en las imagenes.
- **align**: establece la alineación que se usará en la generación de textos en las imagenes. Los posibles valores son "l"para izquierda, ç"para centrado y r"para derecha.
- **opacity**: establece el grado de opacidad que debe usarse en la generación de textos en las imagenes.

- **margin**: establece el margen que debe usarse en la generación de textos en las imagenes. También establece el margen que debe usarse en la prestación over (en este caso, debe especificarse el parámetro margin antes que el parámetro over).
- **rotate / rot**: establece el angulo de rotación que debe usarse en la generación de textos en las imagenes.
- **text / txt / label / caption**: establece el texto que será usado como texto en la generación de la imagen. Este campo debe ser generado con la función base64_encode_url(), la cual además de aplicar el algoritmo base64 sobre el texto, luego escapa dos caracteres que pueden producir errores en la url como són el /z el "+".
- **gray**: aplica un filtro de efecto de escala de grises a la imagen.
- **sep / sepia**: aplica un filtro de efecto de color sepia a la imagen.
- **clr**: aplica una mascara del color especificado por color. Debe ser un color en hexadecimal sin que le preceda el simbolo #.

Notas de esta petición:

- Al generarse las imágenes, estas son guardadas en la cache de RhinOS, de forma que peticiones a una imagen que ya esta generada dan como resultado el contenido del fichero de la cache existente. En la comprobación de la validez de la cache, se comprueban los timestamps de todos los ficheros implicados en la creación, de forma que un cambio del timestamp en la imagen fuente, en la imagen del efecto over o en la fuente empleada en la generación de textos sobre la imagen harán que la cache expire.
- Esta funcion esta protegida contra errores por limitación de memoria en la compresión de salida (en adelante, bug gzhandler).
- Esta función esta protegida contra el bug de IE6 en relación con el envío de imágenes en formato PNG con compresión en la comunicación. Si se detecta un formato PNG con IE6, se cancela la compresión de la salida generada (en adelante, bug ie6png).
- Esta funcionalidad envía las cabeceras para mantener el resultado en la cache del navegador (en adelante, cabeceras de cache para mantener en el navegador y cabeceras de nocache para forzar el no uso de cache en el navegador).
- En algunos servidores cuya configuración de las librerías gráficas no sea del todo correcta, se puede emplear el siguiente define en el fichero config.php del directorio admin para conseguir evitar el uso del ajuste de contraste (llamado USM):

```
define("__PHPTHUMB_USM__",0);
```

Con este define, el sistema omitirá el uso del efecto USM (control de contraste) con lo que en algunos casos, se corregirán los errores que se producen en algunos servidores.

3.5. Otras peticiones - 1

Peticiones a img/, ico/, swf/ y flv/:

Las peticiones a estos directorios únicamente intentan activar la salida con compresión y posteriormente, enviar el contenido del fichero sin aplicar ningún tipo de procesado (esta petición esta protegida contra los bugs gzhandler y ie6png, además de enviar las cabeceras de cache).

Peticiones a rss/, xml/ y xsl/:

Las peticiones a estos directorios hacen que las plantillas solicitadas sean evaluadas por el parser generando la salida (se envían las cabeceras de cache).

Peticiones a htm/:

Estas peticiones, permiten realizar 2 acciones diferentes:

- Si el fichero del template se encuentra en el directorio htm, se evalúa tal como se espera.
- Si el fichero del template se encuentra en el directorio raiz del template (htm/..), se evalúa el fichero del template. Este segundo caso, fuerza que los argumentos de ARGV se desplacen al indice anterior.

En los dos casos, se envían las cabeceras de cache.

Peticiones a print/:

Estas peticiones están orientadas a ofrecer una alternativa de página para la impresión. La idea de esta prestación es poder reciclar todos los contenidos pero pudiendose especificar una plantilla de template secundaria.htm diferente (o la que se defina en la variable $secundaria). El comportamiento de esta petición es idéntica a la petición anterior (htm/), aunque la gran diferencia respecto a la anterior (htm/) es que se empleará la plantilla de secundaria.htm (o la que se defina en la variable $secundaria).

Peticiones a open/ y down/:

Este prefijo indica a RhinOS que se esta solicitando un fichero para abrirlo con el plugin del navegador o forzando su descarga mediante las cabeceras apropiadas. Esta petición se suele emplear al desear poner enlaces a archivos que serán abiertos por otros programas o descargados al ordenador del cliente que realiza la petición.

Notas de esta petición:

- Esta petición esta protegida contra el bug gzhandler.
- Esta petición esta protegida contra el bug de IE con la compresión de archivos que serán procesados por programas externos al navegador.
- Esta petición esta protegida contra el bug de IE con la cabecera Çontent-Disposition: inline".

- Esta petición esta protegida contra el bug de IE con documentos de la suite ofimática de Microsoft y documentos de formato PDF, donde la cabecera "Pragma: no-cache"genera errores.

Peticiones a js/ y json/:
Esta petición evalúa el contenido del fichero de template solicitado.
Notas de esta petición:

- Esta petición esta protegida contra el bug de IE6 con compresión de archivos JavaScript.

Peticiones a css/:
Esta petición envía el contenido del fichero solicitado sin evaluación del parser.
Notas de esta petición:

- Esta petición esta protegida contra el bug de IE6 con compresión de archivos CSS.

3.6. Otras peticiones - 2

Peticiones a cache/:

Estas peticiones implementan funcionalidades de minimización, serialización, cache y compresión para los archivos de tipo CSS y JS.

El uso de esta petición se muestra con el siguiente ejemplo:

```
http://localhost/baseweb/portal/cache?js/codigo1.js,js/codigo2.js,js/codigo3.js
http://localhost/baseweb/portal/cache?css/style1.css,css/style2.css,css/style3.css
```

Con el uso de esta petición, se consigue minimizar las peticiones que el cliente (un navegador generalmente) realiza sobre el servidor. Con este modelo de petición, se obtiene un único fichero que resulta de la minimización del fichero original (dependiendo de cada tipo, CSS o JS, se aplica un algoritmo diferente). Además, se genera un conjunto de ficheros de caches y subcaches que permiten en caso de cambio de cualquier fuente la recomputación del fichero de cache nuevo de la forma más rápida. Esta función usa los timestamps de los ficheros fuentes para determinar la caducidad de la cache (además de estar protegida la petición contra el bug de IE6 con archivos CSS y JS).

Resto de peticiones:

El resto de peticiones, deben actuar sobre plantillas que se encuentren en el directorio templates. Este fichero, será empleado como contenido que se insertará en la plantilla principal (generalmente secundaria.htm, aunque se puede redefinir con la variable $secundaria) cuando se procese la directiva "<!– INCLUDE CONTENT –>"sin ningún argumento adicional, tal como se muestra.

Esta petición realiza la evaluación de la plantilla definida en la variable $secundaria y al procesarse el comando "<!– INCLUDE CONTENT –>", entonces se evaluará la plantilla solicitada por la url (o el nombre del fichero que exista en el array ARGV[0]) generando una salida que sera la concatenación de:

- La porción de la plantilla $secundaria hasta el comando <!-- INCLUDE CONTENT -->.
- El contenido del template solicitado en ARGV[0].
- El resto de la plantilla $secundaria desde el comando <!-- INCLUDE CONTENT -->.

Lo que se pretende con este modelo es poder proporcionar un modelo de plantillas donde se defina en la $secundaria el contenido que aparecerá en todas las salidas y posteriormente, se definan los contenidos personalizados en plantillas independiente, de forma que el resultado esperado ante cualquier petición sea la concatenación de las plantillas tal como se ha mostrado anteriormente.

3.7. El parser de RhinOS

RhinOS, proporciona un parser que permite la evaluación de pseudo-código. Este pseudo-código usa la siguiente sintaxis:

```
<!-- INCLUDE FUNCION ARGUMENTOS -->
```

Tal como se puede observar, el parser usa la notación de comentarios HTML. Cuando se empieza a procesar una plantilla, el parser iterará sobre el pseudo-codigo en busca de patrones regulares que indicarán que tarea debe hacer.

El proceso del parser se ejecuta de la siguiente manera:

- Se busca la primera cadena que contiene <!-- (desde el principio).
- Al encontrarse con una cadena de este tipo, se comprueba si la siguiente palabra es INCLUDE.
- Si es así, se usa como nombre de función la siguiente palabra (por ejemplo,"DBAPP2"), y el resto de palabras como argumento que se le pasará a la función hasta el cierre del tag, ->"(por ejemplo "PRINT nombre").
- La función que se ejecutará sera "dbapp2", que es la palabra "DBAPP2.^{en} minúsculas.
- Esta función recibirá por parámetro la cadena "PRINT nombre".
- Si la función no existe, se intentará cargar con un include el archivo "dbapp2.php".
- Cuando acabe la ejecución de la función dbapp2, se continuará con el resto del pseudo-código restante y se iterará hasta que se alcance el final de la cadena a parsear.

Como se puede ver, el modelo que usa el parser de RhinOS, es:

```
<!-- INCLUDE DBAPP2 PRINT nombre -->
```

Y al ejecutarse el parser, si no existe la función dbapp2, se ejecutará el include para buscar la función. Si tras ejecutarse el include sigue sin existir la función, se generará un error controlado para informar al usuario. Si la función existe, se ejecutará la función dbapp2 recibiendo un único parámetro que contendrá la cadena "PRINT nombre". Esta cadena será procesada por el módulo para ejecutar el comando PRINT sobre el campo nombre.

Con esta explicación, se puede deducir que RhinOS define los módulos de la siguiente manera:

- El modulo debe ser un fichero .php donde el nombre del fichero sea el nombre del módulo en minúsculas.
- Este fichero, debe tener una función con el mismo nombre y que reciba una cadena como parámetro
- Esta cadena (el parámetro) deberá ser procesado por la función para conocer que se le esta pidiendo que realice (en el caso que deseemos que haga varias tareas).

Tal como se ha explicado, RhinOS, define un modelo de definición de funcionalidades que permiten a RhinOS extender sus funcionalidades gracias a los módulos.

A continuación se pone como ejemplo, el código para hacer un módulo helloworld:

```
Contenido del fichero code/helloworld.php
<?php
function helloworld($param) {
echo_buffer("Hello World with arguments: $param");
}
?>
```

Para ejecutar el módulo, se deberá usar un pseudo-código como el siguiente:

```
<!-- INCLUDE HELLOWORLD Argumentos de prueba -->
```

La linea anterior de pseudo-código ejecutará la función helloworld pasando por argumento la cadena "Argumentos de prueba".

3.8. Orden de evaluación del parser de RhinOS

Los comandos que el parser de RhinOS evaluará, deben empezar con la cadena de comentario "<!"seguido de la palabra clave INCLUDE. Esto indicará que el resto de cadena contiene un comando con sus argumentos hasta que finalice el comentario con ">".

El orden en el que el parser evaluará los comandos será:

- Todos los INCLUDE CONTENT (si existen)
- Todos los INCLUDE CACHE (si existen)
- Resto de INCLUDE (si existen)

- Extras del procesado (entidades HTML, entidades UTF y remarcado de buscadores)

Esto permitirá entender como funcionará el parser, pues el proceso de carga de las plantillas con el pseudo-código esta segmentado en tres partes:

1) Resolución de dependencias (INCLUDE CONTENT)

Este proceso se realiza inicialmente cargando la plantilla $secundaria, la cual puede contener comandos INCLUDE CONTENT con o sin argumentos. Estos comandos, son sustituidos por el contenido del fichero al cual hacen referencia. Este proceso se hace hasta que el contenido total de pseudo-código no contiene ningún comando INCLUDE CONTENT. A partir de este punto, se tiene la plantilla a emplear para la generación de la salida preparada con todas las dependencias resueltas.

2) Procesado de los bloques definidos con cache (INCLUDE CACHE)

Este proceso, evaluará las porciones de pseudo-código definidas dentro de los bloques INCLUDE CACHE BEGIN y INCLUDE CACHE END. Este proceso se repetirá para cada bloque detectado en la plantilla.

3) Procesado del resto de comandos (INCLUDE *)

Este proceso, evaluará el resto de comandos no evaluados anteriormente y generará el pseudo-código evaluado en su totalidad. En este punto, el resultado ya se aproxima al resultado que se transmitirá al cliente.

4) Aplicación de los extras sobre la salida generada

Este proceso, obtiene el pseudo-código generado y aplica algunas correcciones como:

- Aplicación de entidades HTML sobre la salida en caracteres extendidos (por ejemplo: acentos o otros caracteres que disponen de entidad HTML propia).
- Corrección sobre algunas entidades UTF-8 que deben ser reparadas (por ejemplo: el símbolo del euro, €).
- Aplicación del algoritmo de remarcado de buscadores que permite sin romper la integridad del resultado, generar un pseudo-código con resaltados de las palabras empleadas en los parámetros GET[q] y GET[p] del SERVER[HTTP_REFERER], si existe.

Al finalizar este proceso, ya se dispone del resultado que se transmitirá hacia el cliente, que se enviará usando algoritmos de compresión (si el cliente lo acepta y se detecta usando las cabeceras que envía al realizar la petición).

3.9. Comandos internos - 1

Comandos PRINT:

A continuación se muestran los comandos que pertenecen al core de RhinOS:

- **PRINT_TODAY_GOOGLE**: Imprime la fecha actual con formato "Y-m-d".
- **PRINT_TITLE**: Imprime el título de la página actual (obtenido a partir de la url).

- **PRINT_PATHINFO**: Imprime la url empleada en la petición (obtenido de $_SERVER["PATHINFO"])
- **PRINT_TAG_BASE**: Imprime el tag <base /> con la url base del portal.
- **PRINT_TAG_RSS_FIREFOX**: Imprime el tag <link /> para generar un feed en los navegadores. La primera palabra del argumento es usada como nombre del fichero xml (por ejemplo, xml/noticias.xml). El resto del argumento es usado como título del feed.
- **PRINT_TAG_FAVICON**: Imprime el tag <link /> para generar un favicon. El argumento es empleado como nombre de la imagen (por ejemplo, img/favicon.ico).
- **PRINT_BASE**: Imprime la url base del portal. Permite indicar como argumento AUTO para detectar si se trata de un puerto seguro o no (https:// o http:*), SSL para forzar la url sobre puerto seguro (https:*) o sin argumento para forzar un puerto no seguro (http://).
- **PRINT_BASE_PARENT**: Identica a la anterior (PRINT_BASE) pero obteniendo la url al directorio padre directamente. Es empleada usualmente para acceder a la base del portal y especificar, por ejemplo, otro idioma.
- **PRINT_LANG**: Imprime el idioma empleado en el portal actual. Su valor puede ser un conjunto de dos letras (es, ca, en, de, ...).
- **PRINT_TIMESTAMP**: Imprime el resultado del comando time(), que es el timestamp unix.
- **PRINT_ARGV_PARAM**: Imprime un valor del array ARGV. Solo lo imprime si existe. El argumento que se le pasa debe ser el número de la posición de ARGV (desde 0 hasta n-1).
- **PRINT_RANDOM**: Imprime un valor pseudo-aleatorio obtenido usando rand, luego uniqid y por último md5.

Comando CONTENT:

CONTENT permite incluir un template. Si se especifica un argumento, este debe ser el nombre del fichero que se desea incluir. Algunos ejemplo de uso del comando CONTENT es:

```
<!-- INCLUDE CONTENT fichero.htm -->
<!-- INCLUDE CONTENT htm/fichero.htm -->
<!-- INCLUDE CONTENT -->
```

En el último ejemplo, como no se especifica ningún argumento, se incluirá el contenido de la variable $content (el valor de esta variable se establece internamente como resultado de ARGV[0], con lo que si se desea cambiar, se debe usar el comando SET_GLOBAL).

Comando SET_GLOBAL:

Este comando permite cambiar el valor de una variable global, de forma que por ejemplo, se puede hacer:

```
<!-- INCLUDE SET_GLOBAL lang es -->
<!-- INCLUDE SET_GLOBAL LANG es -->
<!-- INCLUDE SET GLOBAL _GET[lang] es -->
<!-- INCLUDE SET GLOBAL _GET[LANG] es -->
```

Notas de este comando:

- Este comando permite especificar GET, POST y SESSION en lugar de _GET, _POST y _SESSION:

  ```
  <!-- INCLUDE SET GLOBAL GET[lang] es -->
  <!-- INCLUDE SET GLOBAL GET[LANG] es -->
  ```

  Comando PARSER:

  Evalúa el argumento pasado como código PHP. Se emplea para ejecutar código para inicializar variables, casos donde requiere una programación concreta, ...

  ```
  <!-- INCLUDE PARSER $lang="es"; -->
  <!-- INCLUDE PARSER $LANG="es"; -->
  <!-- INCLUDE PARSER $_GET["lang"]="es"; -->
  <!-- INCLUDE PARSER $_GET["LANG"]="es"; -->
  ```

3.10. Comandos internos - 2

Comando CACHE

RhinOS, proporciona mecanismos para generar cache de los resultados generados del parser, de forma que se recomienda el uso del comando CACHE para obtener un speedup de la ejecución del portal (aunque sea de 1 segundo).

Para hacer uso de este comando, debe usarse la sintaxis:

```
<!-- INCLUDE CACHE OPCION1 OPCION2 -->
```

Donde las opciones serán:

- **OPCION1**:
  - **BEGIN**: Define el inicio de un bloque cuyo resultado generado será guardado en cache hasta que expire. Con el argumento OPCION2 se indica el tiempo de expiración de la misma, de forma que un valor de 300, equivale a 5 minutos (300/60=5). Si se omite el argumento OPCION2, se usará el último valor de tiempo usado en la ultima llamada a <!– INCLUDE CACHE BEGIN XXX –>. Se puedan definir varios valores de tiempo en diferentes llamadas a BEGIN, usando valores elevados para zonas donde no se desee un refresco de la cache rápido y valores pequeños donde sí que se desee un refresco rápido.
  - **END**: Finaliza un bloque que se guardará en cache. No es necesario especificar la OPCION2.

Notas de este comando:

- Este comando, puede emplearse en cualquier zona del template y puede contener el pseudo-código que se desee, pero deben tenerse cuidado con ciertas zonas como formularios donde se esperé que tras el envío, se muestre algún resultado, o pantallas que contienen datos que deben mostrarse en tiempo real. Para estos casos, lo mejor es usar un comando <!– INCLUDE CACHE END > antes de la zona crítica y cuando finalice la misma, usar un comando <! INCLUDE CACHE BEGIN –>. Al no especificar tiempo de expiración, el sistema usará el último valor empleado en el anterior BEGIN donde se ha especificado un valor de tiempo de expiración.

- Es importante entender que los bloques que se guardarán en cache deben ser bloques atómicos, de forma que no existan clausulas IF / ELIF / ELSEIF / ELSE que no se cierren antes de llamar al comando <!– INCLUDE CACHE END –>, pues una porción del pseudo-código quedará fuera de la evaluación y generará errores en la salida no esperados. A continuación se pone un ejemplo de uso correcto del comando CACHE:

```
<!-- INCLUDE CACHE BEGIN 300 -->
Esto se guardará en cache<br/>
<!-- INCLUDE FILTER IF inicio.htm -->
Estoy en la página de inicio<br/>
<!-- INCLUDE FILTER ELSE -->
No estoy en la página de inicio<br/>
<!-- INCLUDE FILTER ENDIF -->
<!-- INCLUDE CACHE END -->
```

  El próximo ejemplo, ilustra un ejemplo de uso incorrecto del comando CACHE:

```
<!-- INCLUDE CACHE BEGIN 300 -->
Esto se guardará en cache<br/>
<!-- INCLUDE FILTER IF inicio.htm -->
Estoy en la página de inicio<br/>
<!-- INCLUDE FILTER ELSE -->
<!-- INCLUDE CACHE END -->
No estoy en la página de inicio<br/>
<!-- INCLUDE FILTER ENDIF -->
```

- Como se puede ver, el control de flujo IF / ELSE / ENDIF esta partido por el comando CACHE, el cual ejecutará en una instancia la primera parte de pseudo-código (el contenido entre el comando CACHE BEGIN y el comando CACHE END) y en otra instancia el resto de pseudo-código, generando una salida que no será la correcta y generará errores en el resultado.

- La cache usa un hash que se genera con la url de la petición, de forma que un cambio de un parámetro en la url de tipo GET, no generará alteración en el hash, con lo que si se usan parámetros por GET, debe controlarse la expiración de la cache.

- Aunque las peticiones sean sobre una misma plantilla, el sistema usará toda la url para el cálculo del hash, con lo que si cambia algún ARGV (generalmente, en algún lugar de la url deberá aparecer como mínimo el identificador del registro mostrado), este generará un hash diferente y en consecuencia, no se produciran colisiones entre las caches generadas.

4. Módulo DBAPP2

El módulo DBAPP2, permite realizar consultas a la base de datos y convertir el procesar el resultado de forma que se pueda generar HTML, JSON, XML o lo que se desee.

Para hacer uso de este módulo, las directivas siempre empezarán por:

```
<!-- INCLUDE DBAPP2 COMANDO CAMPO OPCIONES -->
```

A continuación se listan los comandos agrupados por jerarquía:

- SET y RESET
  - QUERY
    - XML
  - LIMIT
    - OFFSET
      - INTEGER
      - SEQ
      - RND / RAND / RANDOM
    - ARGV_PAG
- INIT
- BEGIN y END
  - DIV_DATA
  - TR_DATA y TD_DATA
  - TR_NULL
  - TD_NULL
  - PAG_PREV, PAG_CURRENT, PAG_NEXT y PAG
  - DINAMICS
  - INCLUDE
- GET
- SET_GLOBAL
- IF / ELIF / ELSEIF / ELSE / ENDIF
- INCLUDE
- PRINT
- IMAGE
- FILE

- VIDEO

Notas de este módulo:

- El módulo DBAPP2 tiene por objetivo el poder definir fragmentos de pseudocódigo y que éste sea empleado en combinación con los resultados obtenidos de las consultas a la base de datos.
- Este comando, permite la sustitución de variables definidas en todos los comandos y opciones pudiendo realizar una gran multitud de características que doten al portal de dinamismo, idiomas y otras prestaciones que posteriormente se explicarán.
- La lista de variables disponibles para sustitución es la siguiente: ARGV, CONFIG, SESSION, GET, POST. ROW estará disponible cuando se haya inicializado una consulta previamente y se este usando un comando BEGIN INCLUDE.
- Para activar el modo DEBUG de este módulo, debe emplearse la palabra clave DEBUG_DBAPP2 en la directiva define DEBUG como se muestra a continuación: define("DEBUG","... DEBUG_DBAPP2 ...");

4.1. Comando SET y RESET

El comando SET establece un valor en el array de tipo static llamado params que será empleado posteriormente en las operaciones del comando INIT para inicializar la consulta y las variables implicadas en el resto del proceso.

El siguiente comando, usualmente es empleado para las siguientes variables del módulo: QUERY, LIMIT, OFFSET, ARGV_PAG y XML.

Comando SET QUERY:

Este comando permite establecer la consulta a la base de datos usando la notación SQL que acepte el SGDB que estemos empleando. También permite emplear las directivas /* */ para poder indicar para que SGBD esta escrita la porción de código SQL:

```
SELECT
id,
/*MYSQL CONCAT(nombre,' ',apellidos) */
/*SQLITE nombre || ' ' || apellidos */ nombre_y_apellidos
FROM tbl_usuarios
```

En este ejemplo, se puede observar como se indica a MYSQL que debe usar la función CONCAT mientras que para SQLITE, se emplea el operador de concatenación del ANSI SQL.

También sería valido indicar una consulta como la siguiente:

```
/*MYSQL SELECT CONCAT(nombre,' ',apellidos) nombre_y_apellidos FROM tbl_usuarios */
/*SQLITE SELECT nombre || ' ' || apellidos nombre_y_apellidos FROM tbl_usuarios */
```

Debe observarse que en definitiva, los dos métodos ofrecen la misma funcionalidad y que el objetivo de este sistema, es poder indicar para una consulta, posibles

variaciones de la misma dependiendo del SGBD que se emplee en cada momento, con lo que se aconseja el uso del primer ejemplo.

La forma práctica de aplicar este comando es la siguiente:

```
<!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_usuarios -->
```

La consulta, puede ser lo compleja que se requiera (este ejemplo, pretende ser una guía de como usar el comando únicamente).

Comando SET XML:

DBAPP2, además de ofrecer la ejecución de consultas contra un SGBD relacional como MySQL o SQLite, también ofrece herramientas para tratar una fuente de código XML como si de una base de datos se tratara.

Para hacer uso de esta prestación, se debe indicar la URL que deberá usarse como fuente del documento XML además de adecuar la consulta (con el comando SET QUERY) usando el lenguaje de consultas XPATH, empleado para obtener nodos de un XML aplicando los filtros que se deseen.

Un ejemplo de uso de este comando sería el siguiente:

```
<!-- INCLUDE DBAPP2 SET XML http://mi.dominio.com/mi/fuente.xml -->
<!-- INCLUDE DBAPP2 SET QUERY //root/data -->
```

Este ejemplo, obtendría un fichero XML de la URL indicada por el SET XML y aplicaría la consulta indicara por SET QUERY de forma que iteraría por cada nodo data que cuelgue del nodo root.

Otro ejemplo de aplicación sería:

```
<!-- INCLUDE DBAPP2 SET XML mi/fuente.xml -->
<!-- INCLUDE DBAPP2 SET QUERY //root/data[@id=2] -->
```

Debe observarse que este ejemplo, usa una URL relativa, con lo que el sistema, añadirá el la ruta de nuestro portal para buscar en el directorio mi el fichero fuente.xml. También, debe observarse que la consulta emplea un filtro donde deberá cumplirse que el nodo data tenga un atributo id cuyo valor para el atributo sea 2.

Comando SET LIMIT:

Este comando indica que debe usarse un límite de resultados en la consulta (que es el resultado de usar el comando LIMIT del ANSI SQL).

A su vez, tiene asociadas varias variables que permiten a LIMIT determinar el OFFSET a usar.

- SET OFFSET SEQ: genera un valor para OFFSET secuencial, de forma que podemos por ejemplo, si definimos una consulta que retorna 15 registros y indicamos un LIMIT de 5, para cada ejecución del comando INIT, el OFFSET se determinará usando la siguiente secuencia: 0, 5, 10, 0, 5, 10, ... Si en lugar de retornar 15 resultados, retornara 12, la secuencia sería: 0, 5, 10, 3, 8, 1, 6, 11, 4, 9, 2, 7, 0, 5, 10, ...
- SET OFFSET RND / RAND / RANDOM: generá un OFFSET pseudoaleatorio de forma que siempre se puedan mostrar el numero indicado en LIMIT de

resultados. Si empleamos el ejemplo anterior (15 resultados en la consulta y LIMIT de 5), el valor de OFFSET oscilará entre 0 y 10 de forma que siempre mostrará 5 resultados. En el caso de que la consulta no retorne un número mayor de registros que el especificado en el LIMIT, el OFFSET será 0. El valor pseudoaleatorio, será siempre obtenido evitando la repetición con la anterior ejecución.

- SET ARGV_PAG: este comando, indica que argumento del array ARGV debe usarse para obtener el número de página, de forma que si se especifica un LIMIT de 5 y se indica ARGV_PAG con valor 1, la siguiente URL (http://mi.dominio.com/portal/noticias/3.htm) generará un offset de 10. El valor por defecto si no se especifica este comando es 1 para ARGV_PAG.

Comando RESET:

Este comando borra una variable establecida anteriormente.

Usualmente es empleada para hacer limpiar las variables LIMIT y OFFSET como en el siguiente ejemplo:

```
<!-- INCLUDE DBAPP2 RESET LIMIT -->
<!-- INCLUDE DBAPP2 RESET OFFSET -->
```

Este comando, permite limpiar todas las variables usando como palabra clave ALL:

```
<!-- INCLUDE DBAPP2 RESET ALL -->
```

4.2. Comando INIT

Este comando inicializa la consulta y define las siguientes variables que posteriormente podrán usarse:

- **NUMROWS**: número de resultados obtenidos con la consulta
- Si se ha especificado un LIMIT:
  - **NUMPAGS**: numero de páginas disponibles con el resultado obtenido de la consulta
  - **USEPAG**: booleano que indica si existe paginación.
  - **ARGV_PAG** si no se ha especificado con valor 1
  - **NUMPAG** con valor 0 si no existe el parámetro especificado o no existe el valor anterior ARGV_PAG que por defecto es 1, o con valor entre 0 y el máximo numero de páginas - 1 si existen más páginas.
  - **OFFSET** si no se ha especificado, será NUMPAG*LIMIT y si se ha especifica como SEQ, RND / RAND / RANDOM, se calculará.
  - **NUM_REG_BEGIN**: el numero de registro (OFFSET + 1)
  - **NUM_REG_END**: el último numero de registro que se mostrará en la página

- **NUM_PAG_CURRENT**: el número de página actual
- **NUM_PAG_FIRST**: siempre es 1
- **NUM_PAG_PREV**: el numero de página anterior (1 como mínimo)
- **NUM_PAG_LAST**: el último número de página posible
- **NUM_PAG_NEXT**: el número de página siguiente (NUM_PAG_LAST como máximo)
- **USE_PAG**: idem que USEPAG
- **USE_PAG_PREV**: booleano que indica si existen páginas anteriores
- **USE_PAG_NEXT**: booleano que indica si existen páginas siguientes
- También serán definidas los resultados obtenidos del primer registro de la ejecución de la consulta sin emplear LIMIT ni OFFSET.

Tras la ejecución de este comando, si únicamente se desea mostrar los resultados del primer resultado, no es necesario el empleo de los comandos BEGIN / END y DIV_DATA / TR_DATA / TD_DATA.

Se recomienda el uso de los resultados sin emplear los comandos BEGIN y END, cuando la consulta sea concisa (sin ambigüedad del resultado que pueda retornar). Por ejemplo, una consulta del tipo SELECT * FROM tbl_xxxxx WHERE id='xxxxx', devolverá un único registro (y no creará ambigüedad). Si se define una consulta que pueda devolver más de un registro y se emplea un comando como SET OFFSET SEQ o SET OFFSET RND, el resultado obtenido no será el esperado. Para ello, deberá emplearse BEGIN y END.

El siguiente fragmento de pseudo-codigo representa la ejecución de una consulta con la representación de los resultados.

```
<!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_productos WHERE imagen!='' -->
<!-- INCLUDE DBAPP2 SET LIMIT 1 -->
<!-- INCLUDE DBAPP2 SET OFFSET 0 -->
<!-- INCLUDE DBAPP2 INIT -->

<!-- INCLUDE DBAPP2 IF NOT NUMROWS -->
NO HAY DATOS<br/>
<!-- INCLUDE DBAPP2 ELSE -->
<!-- INCLUDE DBAPP2 PRINT titulo --><br/>
<!-- INCLUDE DBAPP2 IMAGE imagen --><br/>
<!-- INCLUDE DBAPP2 FILE imagen --><br/>
<!-- INCLUDE DBAPP2 ENDIF -->
```

El pseudo-código anterior, tiene el mismo comportamiento que el siguiente pseudo-código

```
<!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_productos WHERE imagen!='' -->
<!-- INCLUDE DBAPP2 SET LIMIT 1 -->
<!-- INCLUDE DBAPP2 SET OFFSET 0 -->
<!-- INCLUDE DBAPP2 INIT -->

<!-- INCLUDE DBAPP2 IF NOT NUMROWS -->
NO HAY DATOS<br/>
<!-- INCLUDE DBAPP2 ENDIF -->

<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
<!-- INCLUDE DBAPP2 PRINT titulo --><br/>
<!-- INCLUDE DBAPP2 IMAGE imagen --><br/>
<!-- INCLUDE DBAPP2 FILE imagen --><br/>
<!-- INCLUDE DBAPP2 END DIV_DATA -->
```

4.3. Comandos GET y SET_GLOBAL

Comando GET:

Este comando permite a un módulo de programación obtener el valor de una variable que pertenece a la ejecución de DBAPP2 (sólo se empleará desde los módulos de programación y no desde los templates).

Con esta prestación, se puede, por ejemplo, desarrollar un módulo que emplee las prestaciones de DBAPP2 y ejecutar un código PHP como el siguiente:

```
<?php
...
$query="SELECT id,nombre FROM tbl_usuarios WHERE id='123'";
dbapp2("SET QUERY $query");
dbapp2("RESET LIMIT");
dbapp2("RESET OFFSET");
dbapp2("INIT");
nombre=dbapp2("GET nombre");
...
?>
```

Como se puede apreciar, se emplearán los comandos explicados en esta guía para acceder al módulo DBAPP2, establecer la consulta, los parámetros de configuración del módulo y posteriormente ejecutarla. Luego, con el comando GET, se podrán obtener los resultados de esta consulta.

Comando SET_GLOBAL

Establece en una variable el valor de otra variable. Para ello, usa la funcion SET_GLOBAL de RhinOS que permite definir una variable global cualquiera o un valor de los siguientes arrays: GET[], POST[] y SESSION[]. También se pueden acceder a otras estructuras usando notación de PHP como $argv[2].

Para establecer un valor retornado por una consulta tras la ejecución del comando INIT, debe usarse el modelo ROW[campo]. También estarán disponibles todas las variables que se establecen en el comando INIT sin necesidad de usar ningún modificador en el nombre del campo (es decir, sin ROW[]).

4.4. Comando BEGIN y END

Este comando permite indicar porciones de código HTML o XML para que sean empleados según corresponda como plantilla para generar el resultado de la consulta. El comando BEGIN y END,deben usarse conjuntamente tal como se verá en los posteriores ejemplos:

```
<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
<!-- INCLUDE DBAPP2 END DIV_DATA -->

<!-- INCLUDE DBAPP2 BEGIN TR_DATA -->
<!-- INCLUDE DBAPP2 BEGIN TD_DATA -->
<!-- INCLUDE DBAPP2 END TD_DATA -->
<!-- INCLUDE DBAPP2 END TR_DATA -->

<!-- INCLUDE DBAPP2 BEGIN TR_NULL -->
<!-- INCLUDE DBAPP2 END TR_NULL -->

<!-- INCLUDE DBAPP2 BEGIN TD_NULL -->
<!-- INCLUDE DBAPP2 END TD_NULL -->
```

```
<!-- INCLUDE DBAPP2 BEGIN PAG_PREV -->
<!-- INCLUDE DBAPP2 BEGIN PAG_CURRENT -->
<!-- INCLUDE DBAPP2 BEGIN PAG_NEXT -->
<!-- INCLUDE DBAPP2 END PAG -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS tipo -->
<!-- INCLUDE DBAPP2 END DINAMICS [opcional tipo] -->
```

Estos fragmentos de pseudo-código, permiten definir para una consulta, como se desea que se genere el resultado.

4.5. Comando BEGIN y END TR_DATA y TD_DATA

A continuación se mostrarán diversos casos del uso de los tags BEGIN y END para generar una tabla usando HTML:

```
<table>
<!-- INCLUDE DBAPP2 BEGIN TR_DATA -->
<tr>
<!-- INCLUDE DBAPP2 BEGIN TD_DATA -->
<td>
<!-- INCLUDE DBAPP2 PRINT id -->
</td>
<!-- INCLUDE DBAPP2 END TD_DATA -->
</tr>
<!-- INCLUDE DBAPP2 END TR_DATA -->
</table>
```

Este fragmento generará un código donde para cada resultado, abrirá un tag TR y al finalizar el resultado, cerrará el tag TR tal como se muestra.

Si se desea obtener una tabla con un número de nodos TD dentro de cada nodo TR, se puede emplear el comando SET NUM_TD valor donde el valor indicará el número de nodos TD que se desean dentro de cada nodo TR.

Si se desea obtener una tabla con un número mínimo de nodos TR, se puede emplear el comando SET NUM_TR valor donde el valor indicará el número de nodos TR que se desean como mínimo.

4.6. Comando BEGIN y END TR_NULL y TD_NULL

Las dos opciones anteriores son interesantes si deseamos obtener una tabla con un layout concreto, por ejemplo, una tabla con 2 columnas y 3 filas como mínimo. Si el numero de resultados no llena la tabla, podremos obtener una tabla que contenga celdas con valores por defecto.

Para especificar que plantilla debe usarse al generar el resultado de las celdas vacias, se debe emplear el comando BEGIN TR_NULL y BEGIN_TD_NULL, tal como se muestra en el siguiente ejemplo:

```
<!-- INCLUDE DBAPP2 SET LIMIT 2 -->
<!-- INCLUDE DBAPP2 SET NUM_TD 2 -->
<!-- INCLUDE DBAPP2 SET NUM_TR 3 -->
<!-- INCLUDE DBAPP2 BEGIN TR_NULL -->
<tr>
<td>
celda vacia
</td>
```

```
<td>
celda vacia
</td>
</tr>
<!-- INCLUDE DBAPP2 END TR_NULL -->
<!-- INCLUDE DBAPP2 BEGIN TD_NULL -->
<td>
celda vacia
</td>
<!-- INCLUDE DBAPP2 END TD_NULL -->
<table>
<!-- INCLUDE DBAPP2 BEGIN TR_DATA -->
<tr>
<!-- INCLUDE DBAPP2 BEGIN TD_DATA -->
<td>
<!-- INCLUDE DBAPP2 PRINT id -->
</td>
<!-- INCLUDE DBAPP2 END TD_DATA -->
</tr>
<!-- INCLUDE DBAPP2 END TR_DATA -->
</table>
```

4.7. Comando BEGIN y END DIV_DATA

Si deseamos poder generar un código HTML empleando nodos del tipo DIV en lugar de una tabla convencional, DBAPP2 debe usarse de la siguiente manera:

```
<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
<div>
<!-- INCLUDE DBAPP2 PRINT id -->
</div>
<!-- INCLUDE DBAPP2 END DIV_DATA -->
```

De esta manera, cada resultado se generará con un nodo del tipo DIV.

Existen situaciones donde se requiere el poder especificar información extra en los nodos DIV para poder, por ejemplo, usar en los resultados impares un tipo de DIV y en los resultados pares, otro. Supongamos que se requiere generar una lista de nodos DIV pero es necesario que se alternen los resultados entre dos nodos del tipo DIV con atributos diferentes, para ello, existe el comando NEXT_ROW que permite esta prestación:

```
<!-- INCLUDE DBAPP2 SET NUM_TD 2 -->
<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
<div class="tipo1">
<!-- INCLUDE DBAPP2 PRINT id -->
</div>
<!-- INCLUDE DBAPP2 NEXT_ROW -->
<div class="tipo2">
<!-- INCLUDE DBAPP2 PRINT id -->
</div>
<!-- INCLUDE DBAPP2 END DIV_DATA -->
```

El ejemplo anterior, generará una lista de nodos del tipo DIV pero alternará el tipo1 y el tipo2 de forma que obtendremos el resultado deseado.

4.8. Comando BEGIN y END PAG_PREV, PAG_CURRENT y PAG_NEXT

Para generar el resultado de control para el control de páginas, se debe hacer uso de los tags siguientes tal como se muestra en el ejemplo:

```
<!-- INCLUDE DBAPP2 BEGIN PAG_PREV -->
<!-- INCLUDE DBAPP2 PRINT NUM_PAG --> 
<!-- INCLUDE DBAPP2 BEGIN PAG_CURRENT -->
<!-- INCLUDE DBAPP2 PRINT NUM_PAG --> 
<!-- INCLUDE DBAPP2 BEGIN PAG_NEXT -->
<!-- INCLUDE DBAPP2 PRINT NUM_PAG --> 
<!-- INCLUDE DBAPP2 END PAG -->
```

En este ejemplo, el sistema generará el resultado para poder representar la sección de paginación del resultado de la consulta.

Para dotar de enlaces usando nodos del tipo ANCHOR, una posible solución sería (este ejemplo plantea una arquitectura de url semántica usando un patron mi.dominio.com/portal/noticias/pagina.htm)

```
<!-- INCLUDE DBAPP2 BEGIN PAG_PREV -->
<a href="<!-- INCLUDE PRINT_BASE -->noticias/<!-- INCLUDE NUM_PAG -->.htm">
<!-- INCLUDE DBAPP2 PRINT NUM_PAG -->
</a> 
<!-- INCLUDE DBAPP2 BEGIN PAG_CURRENT -->
<!-- INCLUDE DBAPP2 PRINT NUM_PAG --> 
<!-- INCLUDE DBAPP2 BEGIN PAG_NEXT -->
<a href="<!-- INCLUDE PRINT_BASE -->noticias/<!-- INCLUDE NUM_PAG -->.htm">
<!-- INCLUDE DBAPP2 PRINT NUM_PAG -->
</a> 
<!-- INCLUDE DBAPP2 END PAG -->
```

De esta manera, se puede generar el resultado para la representación de una botonera de paginación genérica.

Si se deseá generar únicamente el resultado de paginación cuando exista y evitar mostrar la botonera de paginación en caso de que no sea necesaria (cuando sólo existe una página, por ejemplo), se puede emplear la variable USEPAG (o USE_PAG), la cual controla este aspecto.

```
<!-- INCLUDE DBAPP2 IF USE_PAG -->
... Botonera de paginación ...
<!-- INCLUDE DBAPP2 ENDIF -->
```

4.9. Comando BEGIN y END DINAMICS

Dinamics, es un sistema que permite a RhinOS definir contenidos donde la representación del resultado se define desde el CMS directamente, de forma que para poder generar las plantillas que empleará RhinOS para generar la representación del resultado de la consulta, será necesario usar, por ejemplo, el siguiente código:

```
<!-- INCLUDE DBAPP2 BEGIN DINAMICS titulo -->
Titulo: <!-- INCLUDE DBAPP2 PRINT titulo -->
<!-- INCLUDE DBAPP2 END DINAMICS -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS subtitulo -->
Subtitulo: <!-- INCLUDE DBAPP2 PRINT subtitulo -->
<!-- INCLUDE DBAPP2 END DINAMICS -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS descripcion -->
Descripcion: <!-- INCLUDE DBAPP2 PRINT descripcion -->
<!-- INCLUDE DBAPP2 END DINAMICS -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS foto -->
Foto: <!-- INCLUDE DBAPP2 IMAGE foto -->
```

```
<!-- INCLUDE DBAPP2 END DINAMICS -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS fichero -->
Fichero: <!-- INCLUDE DBAPP2 FILE fichero -->
<!-- INCLUDE DBAPP2 END DINAMICS -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS url -->
URL: <!-- INCLUDE DBAPP2 PRINT url -->
<!-- INCLUDE DBAPP2 END DINAMICS -->

<!-- INCLUDE DBAPP2 BEGIN DINAMICS video -->
Video: <!-- INCLUDE DBAPP2 VIDEO video -->
<!-- INCLUDE DBAPP2 END DINAMICS -->
```

Notas de este comando:

- Los tipos de datos Dinamics, se deben declarar tal como se explica en la documentación del CMS.

4.10. Comando BEGIN y END INCLUDE

Para permitir anidar consultas, existe el comando BEGIN INCLUDE y END INCLUDE que permiten definir donde se inicia un bloque de anidamiento y donde finaliza el mismo. No existe límite de número en los anidamientos de consultas, de forma que podemos tener un pseudo-código como el siguiente:

```
<!-- INCLUDE DBAPP2 SET QUERY SELECT id,nombre FROM tbl_familias -->
<!-- INCLUDE DBAPP2 INIT -->
<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
Familia: <!-- INCLUDE DBAPP2 PRINT nombre -->
<!-- INCLUDE DBAPP2 BEGIN INCLUDE -->
<!-- INCLUDE DBAPP2 SET QUERY SELECT id,nombre FROM tbl_productos WHERE id_familia=ROW[id] -->
<!-- INCLUDE DBAPP2 INIT -->
<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
Producto: <!-- INCLUDE DBAPP2 PRINT nombre -->
<!-- INCLUDE DBAPP2 END DIV_DATA -->
<!-- INCLUDE DBAPP2 END INCLUDE -->
<!-- INCLUDE DBAPP2 END DIV_DATA -->
```

Este comando permite la generación de contenidos como menús de navegación y otras prestaciones necesarias en muchos portales y aplicaciones online.

Notas de este comando:

- Como puede apreciarse en la segunda consulta, se esta aplicando un filtro a la tabla tbl_productos donde el id_familia sea igual al id de la tbl_familias de la consulta padre. Esto permitirá sólo se muestren los productos que pertenecen a la familia que se esta iterando en el bucle exterior.

- Todos los resultados de la consulta exterior, además, estarán disponibles usando la directiva ROW[clave], donde clave es el nombre del campo devuelto por el SGBD.

- Este comando también estará disponible fuera de un bloque BEGIN / END DIV_DATA, TR_DATA y TD_DATA. Esta prestación permite poder emplear, por ejemplo, el siguiente ejemplo:

```
<!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_productos WHERE activado='1' -->
<!-- INCLUDE DBAPP2 SET LIMIT 10 -->
<!-- INCLUDE DBAPP2 SET OFFSET 0 -->
<!-- INCLUDE DBAPP2 INIT -->

<!-- INCLUDE DBAPP2 IF NUMROWS -->
<!-- INCLUDE DBAPP2 BEGIN INCLUDE -->

<!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_productos WHERE activado='1' -->
<!-- INCLUDE DBAPP2 SET LIMIT 10 -->
<!-- INCLUDE DBAPP2 SET OFFSET 0 -->
<!-- INCLUDE DBAPP2 INIT -->

<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
<!-- INCLUDE DBAPP2 PRINT titulo --><br/>
<!-- INCLUDE DBAPP2 END DIV_DATA -->

<!-- INCLUDE DBAPP2 END INCLUDE -->
<!-- INCLUDE DBAPP2 ENDIF -->
```

4.11. Comando IF / ELIF / ELSEIF / ELSE / ENDIF

Estos comandos permiten el control de flujo del resultado generado y los argumentos que pueden recibir los comandos IF / ELIF y ELSEIF son:

Comando IF / ELIF / ELSEIF EVAL condición:

Este comando siempre se ejecuta en el nivel local y debe ser un string con ćodigo PHP que finalice sin el ";". Este comando permite sustitución de variables de forma que se pueden emplear diferentes métodos para obtener el mismo resultado, como el siguiente ejemplo ilustra:

```
<!-- INCLUDE DBAPP2 IF EVAL $argv[1]=="2" -->
```

es igual que:

```
<!-- INCLUDE DBAPP2 IF EVAL \ARGV[1]"=="2" -->
```

El primer ejemplo, es un fragmento de código PHP donde se compara el valor de la variable $argv, indice 1 con el valor del string "2". El segundo ejemplo antes de ejecutarse, realizará la sustitución de variables quedando el código de PHP de la siguiente manera: "X"=="2" donde X será el valor de $argv[1]. La ventaja que tiene el segundo modelo, es que es independiente del lenguaje y estará protegido el código contra errores como el hecho de que, por ejemplo, la variable $argv no esté definido. Se aconseja el uso de "ARGV[1]" antes que emplear directamente $argv[1].

Comando IF / ELIF / ELSEIF NOT condición:

Este comando es igual que el siguiente comando (lease comando IF / ELIF / ELSEIF), añadiendo la diferencia de que la condición será de lógica negada.

Comando IF / ELIF / ELSEIF condición:

Este comando permite la ejecución de la condición en los 2 niveles, dependiendo del ámbito de la variable que interviene en la condición.

Para ello, el interprete, busca si existe la variable en el ámbito actual de ejecución (nivel local) y dependiendo de si existe la variable o no, se ejecuta generando el control de salida esperado o se pospone para una evaluación posterior en el nivel global.

Comando ELSE:

Este comando, no necesita explicación y el resultado será el esperado, es decir, si no se ha evaluado ningún comando IF / ELIF / ELSEIF como cierto, este bloque de pseudo-código se evaluará como cierto activándose el control de salida para que se añada a la salida el fragmento de pseudo-código hasta encontrarse con el comando ENDIF.

Comando ENDIF:

Este comando cierra el control de flujo abierto anteriormente. La no existencia de un ENDIF para una clausula IF / ELIF / ELSEIF / ELSE, generará errores en la salida.

Este comando permite el anidamiento de comandos de la forma que se desee y no tiene límite de número de anidamientos. Se deben cumplir las premisas de programación tradicionales en cuanto al control de flujo, es decir, se deben abrir y cerrar de forma correcta para evitar casos que generarían errores en el sistema de control de flujo.

El siguiente caso ilustra un ejemplo de error del uso de anidamiento:

```
...
<!-- INCLUDE DBAPP2 IF condición -->
...
<!-- INCLUDE DBAPP2 ELIF condición -->
...
<!-- INCLUDE DBAPP2 IF condición_2 -->
...
<!-- INCLUDE DBAPP2 ELSEIF condición -->
...
<!-- INCLUDE DBAPP2 ELSE -->
...
<!-- INCLUDE DBAPP2 ENDIF -->
...
<!-- INCLUDE DBAPP2 ENDIF -->
...
```

Debe observarse que el ENDIF, esta fuera del ámbito del IF condición_2, con lo que se generará un error en la salida producida.

4.12. Comando INCLUDE

El comando INCLUDE, permite a DBAPP2 el control de ejecución de otros módulos usando el control de flujo interno del mismo módulo DBAPP2. Esto implica, por ejemplo, que un comando como INCLUDE DBAPP2 INCLUDE PARSER que se encuentre dentro de un bloque BEGIN / END, será ejecutado en cada iteración. Si se empleará INCLUDE PARSER únicamente, aunque se encuentre dentro de un bloque BEGIN / END del módulo DBAPP2, será ejecutado una única vez (además de ser ejecutado antes del inicio de la generación de la salida) sin ningún tipo de control de ejecución.

El siguiente ejemplo, ilustra como se puede usar el comando INCLUDE para obtener un contador:

```
<!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_productos WHERE activado='1' -->
<!-- INCLUDE DBAPP2 SET LIMIT 10 -->
<!-- INCLUDE DBAPP2 SET OFFSET 0 -->
<!-- INCLUDE DBAPP2 INIT -->
```

```
<!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
<!-- INCLUDE DBAPP2 INCLUDE PARSER
$_GET["contador"]=isset($_GET["contador"])?$_GET["contador"]+1:1; -->
<!-- INCLUDE DBAPP2 PRINT GET[contador] -->: <!-- INCLUDE DBAPP2 PRINT titulo --><br/>
<!-- INCLUDE DBAPP2 END DIV_DATA -->
```

Esto generaría una lista de los 10 títulos obtenidos de la consulta con un número que le precedería y que aumentaría en una unidad en cada iteración.

4.13. Comando PRINT

El comando PRINT, permite la impresión de una variable o campo del resultado de la consulta. Este comando tiene muchas opciones que permitirán realizar un procesado de la salida para adecuarse en cada caso a la necesidad o requerimiento esperado.

La forma de usar este comando es:

```
<!-- INCLUDE DBAPP2 PRINT CAMPO [OPCIONES] -->
```

Como se puede deducir, el CAMPO será el nombre del campo o variable que queremos imprimir y OPCIONES, será una lista separada por espacios que indicarán las opciones que se desean aplicar sobre el resultado del comando PRINT:

- **PREVIEW n**: genera un preview de n caracteres del valor del campo de forma inteligente (quiere decir que no corta palabras, sino que busca la posición más cercana para generar el resultado más esperado).
- **CUT n**: idem que PREVIEW pero cortando en el número exacto de caracteres. Esta prestación eta protegida para el uso de UTF-8 con el empleo de caracteres multibyte.
- **EVAL fn**: aplica un filtro de función sobre el valor del campo, es decir, que ejecutará a fn pasando por argumento el valor del campo a fn y actualizando el mismo valor con el resultado que retorne fn.
- **ENCODE**: genera una cadena con caracteres validos para la generación de URLs semánticas
- **ADD_SLASHES**: aplica addslashes sobre el valor del campo.
- **STRIP_SLASHES**: aplica stripslashes sobre el valor del campo.
- **STRTOLOWER**: aplica strtolower sobre el valor del campo.
- **STRTOUPPER**: aplica strtoupper sobre el valor del campo.
- **UCFIRST**: aplica ucfirst sobre el valor del campo.
- **UCWORDS**: aplica ucwords sobre el valor del campo.
- **REMOVE_EXT**: Elimina la ultima porción del valor usando el ".çomo delimitador.

- **HTML_ENTITY_ENCODE**: aplica htmlentities sobre el valor del campo.
- **HTML_ENTITY_DECODE**: aplica html_entity_decode sobre el valor del campo.
- **UTF8_ENCODE**: aplica utf8_encode sobre el valor del campo.
- **UTF8_DECODE**: aplica utf8_decode sobre el valor del campo.
- **BASE64_ENCODE**: aplica base64_encode sobre el valor del campo.
- **BASE64_DECODE**: aplica base64_decode sobre el valor del campo.
- **BASE64_ENCODE_URL**: idem que BASE64_ENCODE pero sustituye adicionalmente los caracteres $/z$ "+"por "$\_z$ respectivamente de la cadena en base64. Es empleado para la codificación de textos sobre URLs.
- **BASE64_DECODE_URL**: idem que BASE64_DECODE pero sustituye adicionalmente los caracteres "$\_z$ "por $/z$ "+respectivamente de la cadena en base64. Es empleado para la decodificación de textos sobre URLs.
- **XML_ENCODE**: genera valores que validen en documentos XML como sitemaps y feeds RSS
- **NO_CONVERT_DATE**: desactiva la conversión de fecha automática.
- **NO_CONVERT_TIME**: desactiva la conversión de horas automática.
- **CONVERT_DATE_XML**: genera un string con el formato "D, d M Y H:i:s GMT"
- **CONVERT_DATE_TXT**: genera un string con el formato "Dia_sem dia_num de mes_nombre de año".
- **CONVERT_DATE_MONTH_YEAR**: genera un string con el formato "mes_nombre de año".
- **CONVERT_DATE_GOOGLE**: genera un string con el formato "Y-m-d"
- **CONVERT_DATE**: genera un string con el formato "DD/MM/AAAA"
- **CONVERT_TIME**: genera un string con el formato "HH:MM"
- **REPARE_GOOGLE_MAPS**: realiza varias substituciones como quitar los caracteres "\n$z$ "\r", y escapa los caracteres "'z "\ con "\\ 'z "\\ \.
- **REPARE_W3C_BUG**: realiza correcciones sobre el valor producidos por algunos editores WYSIWYG.
- **DELIM**: cadena que se usará como campo delimitador. Requiere de la opción FIELD.
- **FIELD**: número de campo que se desea obtener. Requiere de la opción DELIM.

4.14. Comando IMAGE

El comando IMAGE, permite la generación del pseudo-codigo necesario para la representación de imagenes o de la url de acceso a la misma. La forma de usar este comando es:

```
<!-- INCLUDE DBAPP2 IMAGE CAMPO [OPCIONES] -->
```

Como se puede deducir, el CAMPO será el nombre del campo o variable que queremos usar como imagen y OPCIONES, será una lista separada por espacios que indicarán las opciones que se desean aplicar sobre el resultado del comando IMAGE:

- **WIDTH**: especifica el ancho de la imagen. Requiere de la opción HEIGHT.
- **HEIGHT**: especifica el alto de la imagen. Requiere de la opción WIDTH
- **ID**: especifica el atributo ID del <img /> que se creará.
- **CLASS**: especifica el atributo CLASS del <img /> que se creará. Cada argumento del tipo CLASS se concatena con el valor anterior usando el carácter espacio como separador de forma que usar "CLASS clase1 CLASS clase2" genera un tag .
- **FORMAT**: formato de la imagen que se generará. Por defecto es jpg. Puede ser**: png, gif, tiff, bmp aunque depende de la configuración del servidor y sobretodo, de la librería GD que este instalada con las correspondientes librerías.
- **FAR**: rellena con color de fondo hasta obtener una imagen del tamaño WIDTH y HEIGHT. Este efecto mantiene la relación de aspecto de la imagen. Requiere de las opciones WIDTH y HEIGHT.
- **IAR**: estira la imagen hasta obtener una imagen del tamaño WIDTH y HEIGHT. Este efecto altera la relación de aspecto de la imagen pudiendo generar imágenes con apariencia deformada. Requiere de las opciones WIDTH y HEIGHT.
- **GRAY**: aplica un filtro de efecto de escala de grises a la imagen.
- **SEP / SEPIA**: aplica un filtro de efecto de color sepia a la imagen.
- **TITLE**: establece el campo o valor que se empleará para obtener el texto que se insertará como atributo title en el tag <img />.
- **ALT**: establece el campo o valor que se empleará para obtener el texto que se insertará como atributo alt en el tag <img />.
- **PRINT**: hace que se genere únicamente la url de acceso a la imagen y omite la generación del tag <img />. Es útil cuando se desea conocer la url de una imagen sin que se genere el tag <img /> para poder hacer alguna personalización concreta.

- **RIC n**: redondeo de los bordes de la imagen de radio n. Es útil para obtener imagenes con los bordes redondeados.
- **OVER imagen**: aplica el efecto de overlay usando el argumento adicional imagen. Es útil cuando se desea obtener una imagen cuyo resultado sea más elaborado que un simple efecto como RIC. El argumento adicional imagen, debe indicar el nombre de un fichero que debe encontrarse en el directorio img del template que lo use.
- **BACKGROUNDCOLOR / BACKGROUND / BGCOLOR / BG**: especifica el color de fondo que se usará en los efectos. Debe ser un color en hexadecimal sin que le preceda el simbolo #.
- **FOREGROUNDCOLOR / FOREGROUND / FGCOLOR / FG**: especifica el color de primer plano que se usará en los efectos. Debe ser un color en hexadecimal sin que le preceda el simbolo #.
- **CLR**: aplica una mascara del color especificado por color. Debe ser un color en hexadecimal sin que le preceda el simbolo #.
- **FONT / FN**: establece el nombre de la fuente a usar en la generación de textos en las imagenes. Debe ser un fichero .ttf y no se debe especificar la extensión, debe existir el fichero en el directorio fonts del template que lo use. El valor por defecto es DejaVuSans.
- **SIZE / SZ**: establece el tamaño de la fuente en la generación de textos en las imagenes.
- **ALIGN**: establece la alineación que se usará en la generación de textos en las imagenes. Los posibles valores son "L" para izquierda, "C" para centrado y "R" para derecha.
- **OPACITY**: establece el grado de opacidad que debe usarse en la generación de textos en las imagenes.
- **MARGIN**: establece el margen que debe usarse en la generación de textos en las imagenes. También establece el margen que debe usarse en la prestación OVER (en este caso, debe especificarse el parámetro MARGIN antes que el parámetro OVER).
- **ROTATE / ROT**: establece el angulo de rotación que debe usarse en la generación de textos en las imagenes.
- **TEXT / TXT / LABEL / CAPTION**: establece el campo o valor que se empleará para obtener el texto que será usado como texto en la generación de la imagen. Este campo se empleara como atributo ALT en la generación del tag <img /> y en la generación del nombre de fichero (último argumento de la url semántica y será de la forma resultante a la aplicación de la función ENCODE).

- **EVAL fn**: aplica un filtro de función sobre el valor del campo, es decir, que ejecutará a fn pasando por argumento el valor del campo a fn y actualizando el mismo valor con el resultado que retorne fn.

4.15. Comando FILE

Este comando permite la generación del tag <a />, la generación de la url de acceso al fichero o la generación del tag <img /> del icono del tipo de fichero.

La forma de usar este comando es:

```
<!-- INCLUDE DBAPP2 FILE CAMPO [OPCIONES] -->
```

Las opciones de este comando son:

- **ICON**: establece que en lugar de mostrar el nombre del fichero, se muestre un tag <img /> con el tipo de fichero asociado. Este tag <img /> podra contener las siguientes imágenes que deberán encontrarse en el directorio img del template que lo este usando:
  - **xls.png**: para tipos de archivo con extension xls.
  - **ppt.png**: para tipos de archivo con extension ppt.
  - **doc.png**: para tipos de archivo con extension doc y odt.
  - **pdf.png**: para tipos de archivo con extension pdf.
  - **img.png**: para tipos de archivo con extension bmp, gif, jpg, tif, png, tiff y jpeg.
  - **mdb.png**: para tipos de archivo con extension mdb.
  - **ftp.png**: para tipos de archivo que empiecen por ftp://.
  - **url.png**: para tipos de archivo que empiecen por http:// y https://.
  - **unk.png**: para el resto de archivos
- **TEXT**: establece que se muestre el nombre del archivo (por defecto).
- **OPEN**: establece que se envíen las cabeceras para abrir el fichero (por defecto).
- **DOWN**: establece que se envíen las cabeceras para descargar el fichero (tipo binario).
- **PRINT**: establece que solo se muestre el nombre del fichero (sin tag <a />). Esta opción descarta las anteriores opciones.
- **GET**: establece que se genere únicamente la url del fichero (sin tag <a />). Esta opción sólo permite que funcione las opciones de OPEN y DOWN.

4.16. Comando VIDEO

Este comando permite la generación del tag <object /> para la generación de objetos que representen videos usando Flash.

La forma de usar este comando es:

```
<!-- INCLUDE DBAPP2 VIDEO CAMPO [OPCIONES] -->
```

Las opciones de este comando son:

- **WIDTH**: establece el ancho del objeto (por defecto es de 320px).
- **HEIGHT**: establece el alto del objeto (por defecto es de 240px).
- **SCRIPTACCESS**: establece el atributo scriptaccess (por defecto es always).
- **FULLSCREEN**: establece el atributo fullscreen (por defecto es true).
- **WMODE**: establece el atributo wmode (por defecto es transparent).
- **EMBED**: establece la creación del tag EMBED dentro del tag OBJECT (por defecto es false).
- **PRINT**: únicamente imprime el nombre que se usará para el video (el atributo name).

Actualmente, este comando genera el pseudo-codigo para mostrar un objeto tipo Flash que visualice un reproductor de video de los siguientes proveedores:

- **YouTube**: se detecta por la url www.youtube.com.
- **Vimeo**: se detecta por la url www.vimeo.com.

Dependiendo de cada proveedor, se aplica un patrón regular que genera la url de acceso al movie que debe cargarse, de forma que todos los proveedores, emplean la misma especificación de objeto.

El pseudo-código que se genera es el siguiente para todos los casos:

```
<object type='$type' style='width:${width}px;height:${height}px' data='$name'>
<param name='movie' value='$name'></param>
<param name='allowFullScreen' value='$fullscreen'></param>
<param name='allowscriptaccess' value='$scriptaccess'></param>
<param name='wmode' value='$wmode'></param>
</object>
```

En el caso de haber activado el tag EMBED, se añade esta línea dentro del tag OBJECT:

```
<embed src='$name' type='$type' allowscriptaccess='$scriptaccess'
allowfullscreen='$fullscreen' wmode='$wmode' width='$width' height='$height'></embed>
```

Notas de este comando:

- La variable $name es la url que se calcula para el acceso a la película de Flash.
- La variable $type es "application/x-shockwave-flash".
- Para mejorar el soporte de este comando, es necesario que al detectar cualquier necesidad nueva referente a este comando, se envíe un correo de solicitud a josep.sanz@saltos.net.

5. Módulos adicionales de RhinOS

Esta es la lista de módulos adicionales disponibles:

- Módulo CAPTCHA: Generación de imágenes CAPTCHA.
- Módulo CONFIG: Acceso a variables de configuración.
- Módulo EXTCACHE: Sistema de cache para acelerar los accesos a back-offices.
- Módulo FILES: Acceso a imágenes y ficheros.
- Módulo FILTER: Control de salida mediante condiciones.
- Módulo INTRANET: Control de zonas públicas y privadas mediante identificación.
- Módulo LABELS: Acceso a literales.
- Módulo QUERY: Automatización de consultas como INSERT, UPDATE y DELETE.
- Módulo SEND: Envío de correos con controles adicionales.
- Módulo SESSIONS: Acceso al sistema de sesiones (que se guarda en el SGBD).
- Módulo TIENDA: Procesado de peticiones y gestión de una tienda virtual.

5.1. Módulo CAPTCHA

El módulo CAPTCHA, permite generar una imagen que muestre un número de 4 dígitos usando un fondo que ofusque el trabajo de los robots que intentan hacer submits sobre los formularios de las páginas.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE CAPTCHA ID ARGUMENTOS -->
```

Si no se especifica ningún argumento ni ningún valor para ID, el valor generado se guardará en $_SESSION["captcha"]. Los argumentos que puede recibir son los siguientes.

- **WIDTH**: Ancho de la imagen captcha que se generará (por defecto 150).
- **HEIGHT**: Alto de la imagen captcha que se generará (por defecto 50).
- **LETTER**: Tamaño de la fuente que se usará en las letras del fondo (11 por defecto)
- **NUMBER**: Tamaño de la fuente que se usará en los números del código captcha (28 por defecto).

- **ANGLE**: Angulo de libertad de rotación de las letras y números (10 por defecto).
- **COLOR**: Color que se empleará en los números del código (5C8ED1 por defecto).
- **BGCOLOR**: Color de fondo que se empleará en toda la imagen (C8C8C8 por defecto).
- **FGCOLOR**: Color que se empleará para las letras del fondo (B4B4B4 por defecto).

Generalmente, este comando debe encontrarse en un fichero donde no se generé ningún otro tipo de salida. Su uso frecuentemente es el siguiente:

```
Contenido del fichero templates/htm/get-captcha.htm:
<!-- INCLUDE CAPTCHA -->
```

Este fichero es llamado desde el TAG <img /> que representará la imagen del captcha a entrar en un elemento tipo INPUT que permitirá al usuario la entrada del código de seguridad captcha:

Ejemplo de un formulario que usa captcha:

```
<form action="htm/send-contact" method="post">
Nombre: <input type="text" name="nombre" value="" /><br/>
E-mail: <input type="text" name="email" value="" /><br/>
Comentario: <textarea name="comentario"></textarea><br/>
Código de seguridad: <img src="htm/get-captcha.htm" /><br/>
Entre el código: <input type="text" name="captcha" value="" /><br/>
<input type="submit" value="Enviar" />
</form>
```

Como se puede apreciar, este módulo se concibe como extensión para ser empleado desde otros módulos como SEND o cualquier otro formulario que requiera de control anti robots.

Notas de este comando:

- Para completar un ejemplo, vease el ejemplo del módulo SEND.

Ejemplo de uso:

El siguiente ejemplo ilustra como generar un captcha pudiendo personalizar todos los parámetros desde los argumentos de la llamada GET:

```
<!-- INCLUDE PARSER $_GET["captcha"]=isset($_GET["captcha"])?$_GET["captcha"]:"captcha"; -->
<!-- INCLUDE PARSER $_GET["width"]=isset($_GET["width"])?$_GET["width"]:"150"; -->
<!-- INCLUDE PARSER $_GET["height"]=isset($_GET["height"])?$_GET["height"]:"50"; -->
<!-- INCLUDE PARSER $_GET["letra"]=isset($_GET["letra"])?$_GET["letra"]:"11"; -->
<!-- INCLUDE PARSER $_GET["numero"]=isset($_GET["numero"])?$_GET["numero"]:"28"; -->
<!-- INCLUDE PARSER $_GET["angulo"]=isset($_GET["angulo"])?$_GET["angulo"]:"10"; -->
<!-- INCLUDE PARSER $_GET["color"]=isset($_GET["color"])?$_GET["color"]:"5C8ED1"; -->
<!-- INCLUDE PARSER $_GET["bgcolor"]=isset($_GET["bgcolor"])?$_GET["bgcolor"]:"C8C8C8"; -->
<!-- INCLUDE PARSER $_GET["fgcolor"]=isset($_GET["fgcolor"])?$_GET["fgcolor"]:"B4B4B4"; -->
<!-- INCLUDE CAPTCHA GET[id] WIDTH GET[width] HEIGHT GET[height] LETRA GET[letra] NUMERO GET[numero]
ANGULO GET[angulo] COLOR GET[color] BGCOLOR GET[bgcolor] FGCOLOR GET[fgcolor] -->
```

5.2. Módulo CONFIG

El módulo CONFIG, permite obtener los valores que se han creado en la aplicación de configuración desde el CMS.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE CONFIG COMANDO TAG OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **PRINT**: Idéntico al comando PRINT de DBAPP2 (vease la doc. del módulo DBAPP2).
- **GET**: Retorna el valor del TAG en la configuración (sólo usado desde programación en PHP).
- **SET**: Establece el valor pasado en la configuración del TAG. Este comando permite sustitución usando el sistema de reemplazos de DBAPP2. En los casos donde el parámetro de configuración se ha definido como entero o real, se hace una conversión al formato y se aplican los límites inferiores o superiores definidos (sólo si se han definido para ese TAG).

La aplicación del CMS permite crear valores de configuración con las siguientes características:

- **Parámetro**: se puede definir un nombre del campo de configuración.
- **Tipos**: se pueden definir tipos texto, entero y/o real.
- **Títulos**: se pueden personalizar los títulos que se mostrarán en el CMS para cada campo de configuración.
- **Mínimo**: se puede especificar un valor mínimo para los campos entero y real.
- **Máximo**: se puede especificar un valor máximo para los campos entero y real.

Con estas características, se pueden crear valores de configuración para parametrizar parámetros que luego serán empleados desde los módulos que acepten el uso de CONFIG (como DBAPP2, SEND, ...).

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para la operación de interpretación de comandos con lo que acepta las mismas opciones que el módulo DBAPP2.

5.3. Módulo EXTCACHE

El módulo EXTCACHE, permite emplear el uso de cache para contenidos que se obtienen de fuentes externas como XML. De esta manera, se evitan el uso de llamadas a la fuente de datos y por ejemplo, un sistema basado en RhinOS que use fuentes externas XML (como feeds por ejemplo) con un número elevado de consultas no generará el mismo número de consultas a la fuente XML.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE EXTCACHE ARGUMENTO -->
```

El único parámetro que recibe el módulo debe ser el número de segundos que desea mantenerse la cache hasta que expire. Si no se especifica, el tiempo por defecto es 300 segundos.

Este módulo debe usarse desde un template que no generé salida pues lo que se espera que suceda es que la llamada a este módulo resulte el contenido de la fuente original.

Un ejemplo de uso es el siguiente:

```
Contenido del fichero templates/htm/extcache.htm
<!-- INCLUDE EXTCACHE -->
```

Este fichero, permitirá que desde cualquier lugar se pueda lanzar esta petición:

```
http://localhost/baseweb/templates/htm/extcache.htm?http://www.saltos.net
```

Y petición, en realidad mostrará el contenido que se obtenga de acceder a la url http://www.saltos.net, teniéndola en cache durante el tiempo que se especifique de forma que hasta que transcurran el tiempo de validez de la cache, la llamada anterior devolverá el mismo resultado sin acceder a la fuente original (en el ejemplo es http://www.saltos.net).

5.4. Módulo FILES

El módulo FILES, permite obtener los ficheros e imagenes que se han creado en la aplicación de fotos y documentos desde el CMS.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE FILES TAG OPCIONES -->
```

Este comando no dispone de comandos, pues únicamente, debe especificarse el nombre del TAG que se ha creado en la base de datos y con esta información, el módulo ya identificará si se trata de un fichero o de una foto, generando el resultado esperado.

La aplicación del CMS permite crear fotos y documentos con las siguientes características:

- **Fotos**: indicará que el fichero que se cree es de tipo photo.
- **Fichero**: indicará que el fichero que se cree es de tipo file.

Con estas características, se pueden parametrizar ficheros que no pertenecen a ningún otro tipo de datos ni estan relacionados con ningún contenido que ya exista parametrización en la base de datos.

Un ejemplo típico de uso de este módulo es cuando por ejemplo, se precisa de un fichero de cabecera o cualquier otra foto o fichero que deba de poder ser administrado desde el CMS pero que no pertenecerá a ningún otro modelo de datos (como podrian ser, por ejemplo, noticias, eventos o productos).

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para la operación de interpretación de comandos con lo que acepta las mismas opciones que el módulo DBAPP2 para los tipos de datos IMAGE y FILE.

5.5. Módulo FILTER

El módulo FILTER, permite controlar el control de flujo de la salida generada a partir del valor de las variables de entorno que existen en RhinOS.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE FILTER COMANDO OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **IF / ELIF / ELSEIF**: Permite evaluar las opciones para controlar el flujo de salida. Estos comandos aceptan el parámetro NOT que indica la negación del resto de argumentos que se evaluarán.
- **ELSE**: Activa el control de flujo en caso de que los comandos anteriores (IF / ELIF / ELSEIF) no hayan generado la activación de la salida.
- **ENDIF**: Finaliza el control de flujo de la salida activandola.

Para conseguir que un comando IF / ELIF / ELSEIF active la salida, debe de cumplirse alguna de las condiciones:

- **ARGV[0]**: si el valor de ARGV[0] es igual a alguna opción.
- **GET_LANG**: si el valor devuelto por get_lang() es igual a alguna opción.
- **SESSION**: si se usa alguna opción del tipo SESSION[xxx] y existe $_SESSION["xxx"] cuyo valor se evalúa como cierto.
- **POST**: si se usa alguna opción del tipo POST[xxx] y existe $_POST["xxx"] cuyo valor se evalúa como cierto.
- **GET**: si se usa alguna opción del tipo GET[xxx] y existe $_GET["xxx"] cuyo valor se evalúa como cierto.

Una aplicación de este módulo podría ser:

```
<!-- INCLUDE FILTER IF inicio.htm -->
Estoy en el inicio.htm<br/>
<!-- INCLUDE FILTER ELIF contactar.htm -->
Estoy en el contactar.htm<br/>
<!-- INCLUDE FILTER ELSE -->
Estoy en otra página que me da igual<br/>
<!-- INCLUDE FILTER ENDIF -->
```

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para la operación de interpretación de comandos con lo que acepta las mismas opciones que el módulo DBAPP2.
- Evidentemente, tal como se esperaría en cualquier lenguaje de programación, si un comando IF / ELIF / ELSEIF activa la salida, ningún otro comando IF / ELIF / ELSEIF o ELSE activará la salida hasta que finalice el control de flujo con el comando ENDIF.
- Este módulo permite anidamiento, es decir, que el siguiente ejemplo funcionaría:

```
<!-- INCLUDE FILTER IF inicio.htm -->
Estoy en el inicio.htm<br/>
<!-- INCLUDE FILTER IF es -->
Además estoy en el idioma Castellano<br/>
<!-- INCLUDE FILTER ENDIF -->
<!-- INCLUDE FILTER ELIF contactar.htm -->
Estoy en el contactar.htm<br/>
<!-- INCLUDE FILTER ELSE -->
Estoy en otra página que me da igual<br/>
<!-- INCLUDE FILTER ENDIF -->
```

5.6. Módulo INTRANET

El módulo INTRANET, permite controlar el control de flujo de la salida generada a partir del la consulta de validación que se defina en el módulo.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE INTRANET COMANDO OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **MSG_OK**: Define el mensaje que se usará como mensaje del comando LOGIN y LOGOUT.
- **MSG_KO**: Define el mensaje que se usará como mensaje del comando LOGIN y LOGOUT.
- **MSG_CAPTCHA**: Define el mensaje que se usará como mensaje de error en el captcha.
- **MSG_OUT**: Define el mensaje que se dejará en SESSIONS[error] al ejecutarse un comando LOGOUT.

- **RET_OK**: Define la url de retorno al ejecutarse un comando LOGIN y la validación sea OK. correcta.
- **RET_KO**: Define la url de retorno al ejecutarse un comando LOGIN y la validación sea KO.
- **RET_OUT**: Define la url de retorno al ejecutarse un comando LOGOUT.
- **QUERY**: Permite definir la consulta que se empleará para obtener el registro de validación. Para que el sistema detecte que el sistema es OK en la validación, esta consulta debe devolver únicamente 1 registro. En cualquier otro caso, se evaluará como KO.
- **LOGIN**: Ejecuta el control de validación de acceso. Dejará en SESSION[uid] el id del registro que retorne la consulta de validación, en SESSION[error] el mensaje de MSG_OK o MSG_KO dependiendo del resultado de la validación y en SESSION[has_error] un valor 0 en caso de no error y un valor 1 en caso de error. Este comando borra la variable CAPTCHA definida en el comando CAPTCHA.
- **LOGOUT**: Ejecuta el control de validación de acceso. Dejará en SESSION[error] el mensaje de MSG_OUT y en SESSION[has_error] el valor 0, además de dejar la variable SESSION[uid] con valor 0.
- **OK**: Define el inicio de un bloque que debe mostrarse en la salida en caso de validación OK.
- **KO**: Define el inicio de un bloque que debe mostrarse en la salida en caso de validación KO.
- **ALL**: Finaliza la definición de bloques que debe controlar este módulo.
- **RETURN**: , Se retornará a la url definida en RET_OK, RET_KO o RET_OUT, dependiendo del estado de la variable de SESSION[uid] y SESSION[has_error]. Se usará HTTP_REFERER en caso de no estar definida alguna de la variable RET_OK, RET_KO o RET_OUT o si se pasa el argumento REFERER al comando RETURN.
- **CAPTCHA**: Define la variable de sesión que se empleará para la validación del CAPTCHA. En caso de no definirse, no se empleará validación del CAPTCHA. La comprobación del CAPTCHA, solo se efectuará en el comando LOGIN.

Una aplicación de este módulo podría ser:

```
Estoy fuera del bloque de validación<br/>
<!-- INCLUDE INTRANET QUERY SELECT id,nombre FROM tbl_usuarios WHERE user='SESSION[user]' AND pass='SESSION[pass]' -->
<!-- INCLUDE INTRANET OK -->
Estoy validado<br/>
<!-- INCLUDE INTRANET KO -->
No estoy validado<br/>
<!-- INCLUDE INTRANET ALL -->
Estoy fuera del bloque de validación<br/>
```

Previamente, se deberá de haber dejado en el array SESSIONS los valores de "user" y "pass" para que funcione el ejemplo anterior. También se deberán borrar estos valores para conseguir la acción de desconexión (comando LOGOUT). Unos ejemplos de template usados en los procesos de LOGIN y LOGOUT pueden ser los siguientes:

```
Contenido del fichero htm/intranet-login.htm
<!-- INCLUDE SESSIONS SET has_error 0 -->
<!-- INCLUDE SESSIONS SET user GETPARAM -->
<!-- INCLUDE SESSIONS SET pass GETPARAM -->
<!-- INCLUDE INTRANET MSG_KO Bienvenido. -->
<!-- INCLUDE INTRANET MSG_KO Acceso denegado. -->
<!-- INCLUDE INTRANET RET_OK intranet.htm -->
<!-- INCLUDE INTRANET RET_KO intranet.htm -->
<!-- INCLUDE INTRANET LOGIN -->
<!-- INCLUDE INTRANET RETURN -->

Contenido del fichero htm/intranet-logout.htm
<!-- INCLUDE SESSIONS SET has_error 0 -->
<!-- INCLUDE SESSIONS SET user NULL -->
<!-- INCLUDE SESSIONS SET pass NULL -->
<!-- INCLUDE INTRANET MSG_OUT Hasta luego. -->
<!-- INCLUDE INTRANET RET_OUT intranet.htm -->
<!-- INCLUDE INTRANET LOGOUT -->
<!-- INCLUDE INTRANET RETURN -->
```

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para la operación de interpretación de comandos con lo que acepta las mismas opciones que el módulo DBAPP2. Este módulo permite anidamiento, es decir, que el siguiente ejemplo funcionaría:

  ```
  <!-- INCLUDE INTRANET OK -->
  Estoy validado<br/>
  <!-- INCLUDE DBAPP2 SET QUERY SELECT * FROM tbl_productos -->
  <!-- INCLUDE DBAPP2 INIT -->
  <!-- INCLUDE DBAPP2 BEGIN DIV_DATA -->
  Producto: <!-- INCLUDE DBAPP2 PRINT nombre --><br/>
  <!-- INCLUDE DBAPP2 END DIV_DATA -->
  <!-- INCLUDE INTRANET KO -->
  No estoy validado<br/>
  <!-- INCLUDE INTRANET ALL -->
  ```

- Este módulo permite la integración en aplicaciones que usen AJAX para las tareas de recarga de contenidos. Para ello, tras la ejecución de los comandos LOGIN y LOGOUT, el sistema enviará el pseudo-código que permitirá capturar el evento del contenido a cargar definiendo la siguiente función:

  ```
  <script type="text/javascript">
  function _intranet_new_location2(param,has_error,error) {
  ...
  }
  </script>
  ```

  El parámetro que recibe la función es la url de retorno que se empleará, un entero (1=true, 0=false) que indica si ha existido un error y el mensaje correspondiente. En caso de no estar definida esta función, el sistema ejecutará el siguiente pseudo-código:

```
<script type="text/javascript">
function _intranet_new_location(param) {
...
}
</script>
```

Y si esta función tampoco existe, se ejecuta como último caso el siguiente pseudo-código:

```
<script type="text/javascript">
window.location.href=param;
</script>
```

- Los comandos LOGIN y LOGOUT solo se ejecutarán si no existe la variable SESSION[has_error] con valor activado. Esto sucede así para evitar la ejecución en caso de que una llamada previa a algún módulo (inclusive éste) que genere algún error rompa la integridad del proceso tal como se espera. Es responsabilidad del integrador, controlar después de cada ejecución el correcto estado de retorno de SESSION[has_error].

5.7. Módulo LABELS

El módulo LABELS, permite obtener los literales que se han creado en la aplicación de literales desde el CMS.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE LABELS COMANDO TAG OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **PRINT**: Idéntico al comando PRINT de DBAPP2 (vease la doc. del módulo DBAPP2).
- **GET**: Retorna el valor del literal solicitado (sólo usado desde programación en PHP).
- **IMAGE**: Idéntico al comando IMAGE de DBAPP2 (vease la doc. del módulo DBAPP2). Este comando permite generar una imagen que muestre el texto guardado en un label usando una imagen que es pasada como argumento como fondo de la imagen. A continuación se muestra un ejemplo de uso:

  ```
  <!-- INCLUDE LABELS IMAGE TAG imagen.png PARAMS -->
  ```

  Donde:

  - **TAG**: Es el nombre del label (tag) que se desea usar.
  - **imagen.png**: Es la imagen que se empleará como fondo de la imagen y debe existir en el directorio img en el template que se este empleando.

- **PARAMS**: Lista de argumentos adicionales que serán procesados por el comando IMAGE de DBAPP2 para aplicar correcciones en la imagen y/o en el pseudo-código generado.

- **DEBUG**: Permite activar el modo debug del módulo labels en la sesión que se ejecute. Esto permite que el portal pueda saltar al modo debug de labels forzando que en todas las peticiones al módulo labels, este retorne el valor del TAG en lugar del valor esperado del propio label. A continuación se muestra un ejemplo de uso:

```
Contenido del fichero htm/labels_debug_session_on.htm
<!-- INCLUDE LABELS DEBUG 1 -->
<script type='text/javascript'>
window.location.href='../inicio.htm';
</script>


Contenido del fichero htm/labels_debug_session_off.htm
<!-- INCLUDE LABELS DEBUG 0 -->
<script type='text/javascript'>
window.location.href='../inicio.htm';
</script>
```

  Con el ejemplo anterior, se estarían publicando 2 urls para poder acceder a la activación y desactivación del módulo labels, dando como resultado que tras la llamada al primer htm (DEBUG 1), el portal mostraría todos los TAGS de los labels, en lugar del valor esperado de base de datos. Para desactivar esta prestación, bastará con hacer una petición al segundo htm (DEBUG 0), con lo que el portal, volverá a mostrar todos los valores en los labels existentes.

La aplicación del CMS permite crear literales con las siguientes características:

- **Secciones**: se agrupan por secciones para una mayor facilidad de administración.

- **Idiomas**: pueden existir diferentes idiomas para poder proporcionar sites multi-idioma.

- **Tipos**: Los campos pueden definirse de tipo text o textarea.

Con estas características, se pueden crear secciones para cada página y el usuario, dispondrá de un interface que le simplificará el trabajo al mostrar la información segmentada por secciones e idiomas.

Los tipos, permiten que se muestre en el CMS de administración, campos de tipo text (una sola línea) o campos de tipo textarea (usando el editor CKEditor). La diferencia técnica entre estos tipos de datos es que el tipo text guarda únicamente un texto tal como se escribe, mientras que el tipo textarea, generará un pseudo-código HMTL que será guardado en la base de datos tal como se obtenga.

Este comando puede ser empleado con o sin secciones, pero dependiendo si se ha activado la prestación de secciones o no, podremos emplear una notación donde se indique el TAG únicamente o SECCION/TAG para el acceso a un literal:

```
<!-- INCLUDE LABELS PRINT MI_SECCION/MI_LITERAL -->
```

Este fragmento de pseudo-codigo obtendrá el literal MI_LITERAL de la sección MI_SECCION. Si se omite el dato de MI_SECCION, el módulo retornará el valor del primer literal que encuentre (en la sección que sea) cuyo TAG sea MI_LITERAL.

En el caso de no tener activada la prestación de secciones, puede emplearse de la siguiente manera:

```
<!-- INCLUDE LABELS PRINT MI_LITERAL -->
```

De esta manera, si existen o no secciones, pero si solo existe un literal con el TAG MI_LITERAL, el módulo nos representará su valor en la salida generada.

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para la operación de interpretación de comandos con lo que acepta las mismas opciones que el módulo DBAPP2.

5.8. Módulo QUERY

El módulo QUERY, permite generar consultas al SGBD del tipo SELECT, INSERT, UPDATE y DELETE.

Con este módulo, se pueden automatizar por ejemplo, el control de accesos en el módulo INTRANET, guardar los envíos del módulo SEND en el SGBD, actualizar formularios con datos del perfil del usuario y todo aquello que se desee.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE QUERY COMANDO OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **MSG_UNIQUE**: Define el mensaje que se usará como mensaje de error cuando se ejecute un comando INSERT o un comando UPDATE y un campo definido con UNIQUE tenga un valor duplicado.
- **MSG_NEEDED**: Define el mensaje que se usará como mensaje de error cuando se ejecute un comando INSERT o un comando UPDATE y un campo definido con NEEDED no tenga valor definido.
- **MSG_CAPTCHA**: Define el mensaje que se usará como mensaje de error en el captcha.
- **TABLE**: Permite definir el nombre de la tabla que se empleará en la consulta.
- **FIELDS**: Permite definir la lista de campos que se empleará en la consulta. Sólo se emplearán en los comandos SELECT, INSERT y UPDATE.
- **TYPES**: Permite definir para la lista de FIELDS, los tipos de datos que se deben usar. Los tipos permitidos son INTEGER, FLOAT y TEXT.

- **NEEDEDS**: Permite definir la lista de campos que serán necesarios en la consulta. Sólo se emplearán en los comandos INSERT y UPDATE.
- **UNIQUES**: Permite definir la lista de campos que deberán ser únicos en la consulta. Sólo se emplearán en los comandos INSERT y UPDATE.
- **KEY**: Permite definir el nombre del campo que se empleará en el filtro. Sólo se emplearán en los comandos SELECT, UPDATE y DELETE.
- **VALUE**: Permite definir el valor que se empleará en el filtro. Sólo se emplearán en los comandos SELECT, UPDATE y DELETE.
- **SELECT**: Ejecuta el comando SELECT en el SGBD. Si se emplea el argumento adicional PRINT, el sistema imprimirá en la salida generada la consulta SQL que se emplee.
- **INSERT**: Ejecuta el comando INSERT en el SGBD. Si se emplea el argumento adicional PRINT, el sistema imprimirá en la salida generada la consulta SQL que se emplee.
- **UPDATE**: Ejecuta el comando UPDATE en el SGBD. Si se emplea el argumento adicional PRINT, el sistema imprimirá en la salida generada la consulta SQL que se emplee.
- **DELETE**: Ejecuta el comando DELETE en el SGBD. Si se emplea el argumento adicional PRINT, el sistema imprimirá en la salida generada la consulta SQL que se emplee.
- **PRINT**: Imprime en la salida generada el valor del campo obtenido en la última ejecución de SELECT.
- **CAPTCHA**: Define la variable de sesión que se empleará para la validación del CAPTCHA. En caso de no definirse, no se empleará validación del CAPTCHA. La comprobación del CAPTCHA, se efectuará en caso de definirse en los comandos SELECT, INSERT, UPDATE y DELETE. Tras la ejecución de uno de estos comandos, la variable CAPTCHA quedará borrada.
- **SET_GLOBAL**: Establece en una variable el valor de otra variable. Para ello, usa la funcion SET_GLOBAL de RhinOS que permite definir una variable global cualquiera o un valor de los siguientes arrays: GET[], POST[] y SESSION[]. También se pueden acceder a otras estructuras usando notación de PHP como $argv[2].

Notas de este módulo:

- Este comando emplea el módulo DBAPP2, con lo que los comandos emplearán sustituciones, tal como se documenta en el módulo DBAPP2.
- Los resultados de la última ejecución del comando SELECT, estarán disponibles para sustitución usando la clave ROW[campo].

- Después de cada ejecución del comando INSERT, estará disponible para su uso la variable ROW[LAST_INSERT_ID] que contendrá el ID del registro creado en el comando INSERT.
- Los valores definidos con los comandos TABLE, FIELDS, TYPES, UNIQUES, NEEDEDS, KEY, VALUE y CAPTCHA son borrados después de cada ejecución de los comandos SELECT, INSERT, UPDATE y DELETE.
- Los comandos SELECT, INSERT, UPDATE y DELETE sólo se ejecutarán si no existe la variable SESSION[has_error] con valor activado. Esto sucede así para evitar la ejecución de consultas a la base de datos en caso de que una llamada previa a algún módulo (inclusive éste) que genere algún error rompa la integridad del proceso tal como se espera. Es responsabilidad del integrador, controlar después de cada ejecución el correcto estado de retorno de SESSION[has_error].

Ejemplo de uso:

```
<!-- INCLUDE SESSIONS SET has_error 0 -->

<!-- INCLUDE SET_GLOBAL GET[titulo] Mi titulo sin repeticiones -->
<!-- INCLUDE SET_GLOBAL GET[codigo] pepetito -->

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY FIELDS MAX(id) maximo -->
<!-- INCLUDE QUERY SELECT -->
<!-- INCLUDE QUERY PRINT maximo --><br/>

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY FIELDS id,codigo,titulo,imagen,imagen_file,importe -->
<!-- INCLUDE QUERY NEEDEDS titulo -->
<!-- INCLUDE QUERY UNIQUES codigo -->
<!-- INCLUDE QUERY INSERT -->

<!-- INCLUDE QUERY PRINT LAST_INSERT_ID --><br/>

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY FIELDS MAX(id) maximo -->
<!-- INCLUDE QUERY SELECT -->
<!-- INCLUDE QUERY PRINT maximo --><br/>

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY FIELDS id,codigo,titulo,imagen,imagen_file,importe -->
<!-- INCLUDE QUERY NEEDEDS titulo -->
<!-- INCLUDE QUERY UNIQUES codigo -->
<!-- INCLUDE QUERY KEY id -->
<!-- INCLUDE QUERY VALUE ROW[maximo] -->
<!-- INCLUDE QUERY UPDATE -->

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY FIELDS MAX(id) maximo -->
<!-- INCLUDE QUERY SELECT -->
<!-- INCLUDE QUERY PRINT maximo --><br/>

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY KEY id -->
<!-- INCLUDE QUERY VALUE ROW[maximo] -->
<!-- INCLUDE QUERY DELETE -->

<!-- INCLUDE QUERY TABLE tbl_productos -->
<!-- INCLUDE QUERY FIELDS MAX(id) maximo -->
```

```
<!-- INCLUDE QUERY SELECT -->
<!-- INCLUDE QUERY PRINT maximo --><br/>

<!-- INCLUDE QUERY SET_GLOBAL GET[maximo] ROW[maximo] -->
```

5.9. Módulo SEND

El módulo SEND, permite enviar correos a múltiples destinatarios. Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE SEND COMANDO OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **MSG_OK**: Define el mensaje que se usará como mensaje de envío correcto.
- **MSG_KO**: Define el mensaje que se usará como mensaje de envío erróneo.
- **MSG_CAPTCHA**: Define el mensaje que se usará como mensaje de error en el captcha.
- **MSG_NEEDED**: Define el mensaje que se usará como mensaje de error cuando se realice un envío y un campo definido con NEEDED no tenga valor definido.
- **RET_OK**: Define la url de retorno para un envío correcto.
- **RET_KO**: Define la url de retorno para un envío erroneo.
- **FROM**: Define el email a usar como FROM.
- **FROMNAME**: Define el texto a usar como FROMNAME.
- **TO**: Define el email a usar como destinatario. Este comando permite llamadas multiples de forma que n llamadas al comando TO, generaría una lista de n destinatarios del correo. Se controla el caso de repeticiones descartando los destinatarios existentes.
- **SUBJECT**: Define el SUBJECT a usar en el correo.
- **FIELDS**: Define la lista separada por comas de los campos que llegarán por POST.
- **NAMES**: Define la lista de los nombres de los campos que llegarán por POST.
- **TYPES**: Define la lista de los tipos de los campos que llegarán por POST. Los posibles tipos que se pueden emplear son:
    - **TEXT**: Define un campo de tipo texto. Se representará en el correo como una caja de texto de una línea.

- **TEXTAREA**: Define un campo de tipo textarea. Se representará como una caja de texto de varias lineas donde el nombre se mostrará en una caja centrada y el valor se mostrará en otra caja usando todo el ancho del report generado. Si no se define nombre (NAMES) para éste campo, es omitirá la representación de la caja que debe contener el nombre.
- **MAIL**: Define un campo de tipo correo electrónico. Se representará como una caja de texto de una línea siendo el texto mostrado un enlace al mismo valor con el prefijo "mailto:".
- **LINK**: Define un campo de tipo enlace. Se representará como una caja de texto de una línea siendo el texto mostrado un enlace al mismo valor.
- **FILE**: Define un campo de tipo fichero. Se presentará como una caja de texto de una línea y se añadirá un attachment en el correo con el fichero recibido en el array FILES.

- **NEEDEDS**: Permite definir la lista de campos que serán necesarios para realizar el envío.
- **BACKUP**: Permite usar una variable de sesión para guardar o recuperar los datos del POST:
- **SET**: Guarda en una variable de sesión todos los valores del POST.
- **RESTORE**: Carga en el POST los valores guardados previamente con el comando SET.
- **RETURN**: Se retornará a la url definida en RET_OK o RET_KO, dependiendo del estado de la variable de SESSION[has_error]. Se usará HTTP_REFERER en caso de no estar definida alguna de la variable RET_OK o RET_KO o si se pasa el argumento REFERER al comando RETURN.
- **CAPTCHA**: Define la variable de sesión que se empleará para la validación del CAPTCHA. En caso de no definirse, no se empleará validación del CAPTCHA.
- **HOST**: Define el servidor SMTP que se empleará para el envío. En caso de no especificarse, se usará el servidor instalado en el propio servidor (sendmail la gran mayoría de casos)
- **USER**: Define el usuario que se empleará en el servidor SMTP (sólo necesario cuando se especifica el comando HOST).
- **PASS**: Define la contraseña que se empleará en el servidor SMTP (sólo necesario cuando se especifica el comando HOST).
- **COLOR**: Define el color que se empleará en la generación del report que se enviará al destinatario del envío. Estos valores pueden ser BGHEADER, FGHEADER, BGREPORT y FGREPORT.
- **PRINT**: Permite obtener el resultado del cuerpo del mensaje que se enviará sin realizar el envío.

- Si no se especifica comando, este módulo realiza la acción de enviar el correo.

Los siguientes caracteres en los campos FROM y TO son protegidos de forma que se pueden esperar las siguientes sustituciones:

- **[AT]**: es reemplazado por "@"
- **[DOT]**: es reemplazado por "."

También permite el uso de valores obtenidos de los módulos labels y config en los siguientes campos: MSG_OK, MSG_KO, MSG_CAPTCHA, RET_OK, RET_KO, FROM, FROMNAME, TO y SUBJECT.

El campo TO, adicionalmente, permite sustituciones compatibles con DBAPP2 (son las del tipo GET[], POST[], SESSION[], CONFIG[] y ARGV[]).

Para emplear la prestación anterior, bastaría con usar los siguientes comandos

```
<!-- INCLUDE SEND MSG_OK LABEL MENSAJE_ENVIO_OK -->
<!-- INCLUDE SEND MSG_OK LABELS MENSAJE_ENVIO_OK -->
<!-- INCLUDE SEND TO CONFIG DESTINATARIO_FORMULARIO_CONTACTO -->
```

En el momento del envío, se producen las siguientes comprobaciones y acciones:

- Los datos que llegan por POST son guardados en el array BACKUP de la sesión actual.
- Si se ha especificado un valor para la variable CAPTCHA, se realizará la validación de seguridad:
- Si los valores empleados en la validación del captcha no tienen longitud de 4 carácteres o son diferentes, la validación fallará.
- En caso de fallar la validación, se establece $_SESSION["error"] el valor del mensaje MSG_CAPTCHA, se establece también $_SESSION["has_error"] el valor '1'.
- Se genera el pseudo-código HTML que será empleado como cuerpo del correo electrónico.
- Se procede al envío de los correos a cada destinatario definido en el TO.
- En caso de no producirse ningún error, se establece el valor MSG_OK en $_SESSION["error"], se establece también en $_SESSION["has_error"] el valor '0'.
- En caso de producirse algún error, se establece en $_SESSION["error"] el valor del mensaje MSG_KO, se establece también en $_SESSION["has_error"] el valor '1'.
- Se borran la lista de destinatarios (comando TO) y el valor del captcha (comando CAPTCHA).

A continuación se muestra un posible ejemplo de uso de este módulo para enviar un formulario de contacto:

```
<!-- INCLUDE SESSIONS SET has_error 0 -->
<!-- INCLUDE SEND MSG_OK El e-mail se ha enviado correctamente. -->
<!-- INCLUDE SEND MSG_KO Se ha producido un error en el envío. -->
<!-- INCLUDE SEND MSG_CAPTCHA Código de seguridad inválido. -->
<!-- INCLUDE SEND MSG_NEEDED Falta algún campo obligatorio -->
<!-- INCLUDE SEND FROM info[at]saltos[dot]net -->
<!-- INCLUDE SEND FROMNAME www.saltos.net -->
<!-- INCLUDE SEND TO info[at]saltos[dot]net -->
<!-- INCLUDE SEND SUBJECT Formulario de contacto -->
<!-- INCLUDE SEND FIELDS nombre,email,comentario -->
<!-- INCLUDE SEND NAMES Nombre,E-mail,Comentario -->
<!-- INCLUDE SEND TYPES text,text,textarea -->
<!-- INCLUDE SEND NEEDEDS nombre,email -->
<!-- INCLUDE SEND CAPTCHA captcha -->
<!-- INCLUDE SEND COLOR BGHEADER #336699 -->
<!-- INCLUDE SEND -->
<!-- INCLUDE SEND RETURN -->
```

Notas de este módulo:

- Este módulo permite la integración en aplicaciones que usen AJAX para las tareas de recarga de contenidos. Para ello, tras la ejecución del comando SEND RETURN, el sistema enviará el pseudo-código que permitirá capturar el evento del contenido a cargar definiendo la siguiente función:

  ```
  <script type="text/javascript">
  function _send_new_location2(param,has_error,error) {
  ...
  }
  </script>
  ```

  El parámetro que recibe la función es la url de retorno que se empleará, un entero (1=true, 0=false) que indica si ha existido un error y el mensaje correspondiente. En caso de no estar definida esta función, el sistema ejecutará el siguiente pseudo-código:

  ```
  <script type="text/javascript">
  function _send_new_location(param) {
  ...
  }
  </script>
  ```

  Y si esta función tampoco existe, se ejecuta como último caso el siguiente pseudo-código:

  ```
  <script type="text/javascript">
  window.location.href=param;
  </script>
  ```

- El comando INCLUDE SEND sin argumentos sólo se ejecutarán si no existe la variable SESSION[has_error] con valor activado. Esto sucede así para evitar el envío de emails en caso de que una llamada previa a algún módulo (inclusive éste) que genere algún error rompa la integridad del proceso tal como se espera. Es responsabilidad del integrador, controlar después de cada ejecución el correcto estado de retorno de SESSION[has_error].

- Este módulo puede usar un DEFINE que permite forzar la validación de las direcciones de correo para empleos meramente depurativos:

```
define("__FORCE_ISMAIL__",true);
```

5.10. Módulo SESSIONS

El módulo SESSION, permite usar las sesiones que implementa RhinOS sobre el SGBD.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE SESSIONS COMANDO OPCIONES -->
```

Este comando dispone de los siguientes comandos:

- **SET**: Establece un valor en una variable del array SESSIONS. Este comando permite argumentos adicionales como:
- **NULL**: Para forzar establecer un valor vacio (“”).
- **GETPARAM**: Establece el parámetro que exista en el array GET o POST.
- **GET**: Retorna el valor de la clave solicitada (sólo usado desde programación en PHP).
- **CLOSE**: Fuerza la llamada para cerrar la sesión (no es necesario usarlo para disponer de sesiones y sólo debe usarse desde un módulo conociendo su comportamiento).
- **INIT**: Fuerza la llamada para iniciar la sesión (no es necesario usarlo para disponer de sesiones y sólo debe usarse desde un módulo conociendo su comportamiento).
- **SAVE**: Fuerza la llamada para guardar los cambios en la sesión (no es necesario usarlo para disponer de sesiones y sólo debe usarse desde un módulo conociendo su comportamiento).
- **PRINT**: Idéntico al comando PRINT de DBAPP2 (vease la doc. del módulo DBAPP2). La diferencia con el comando PRINT de DBAPP2 es que no es necesario usar SESSION[clave] para imprimir el valor de clave. Basta con usar <!– INCLUDE SESSIONS PRINT clave –> para imprimir el valor de clave del array SESSIONS.

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para el comando PRINT con lo que acepta las mismas opciones que el módulo DBAPP2.

- Este comando, si es usado desde un módulo, puede emplearse para establecer el valor de un array completo como variable de sesión. Para ello, puede pasarse un segundo argumento a la función sessions() que contenga el array a establecer. A continuación se muestra un ejemplo de uso en un módulo de programación:

```
<?php
function mimodulo($param) {
include_once(\sessions.php");
$lista=array(\valor1","valor2","valor3");
sessions(\SET valores",$lista);
}
?>
```

5.11. Módulo TIENDA

El módulo TIENDA, permite definir una tienda online con todas las funcionalidades que se esperan un portal con tienda online.

Para hacer uso desde los templates de esta prestación, debe usarse:

```
<!-- INCLUDE TIENDA COMANDO OPCIONES -->
```

Este comando, dispone de los siguientes comandos:

- BEGIN y END
  - HEADER / PRODUCT / FOOTER
  - VOID
- QUERY
  - PRODUCT
  - PRICE / UNITS / DISCOUNT / TAX / SHIPPING
  - MAIN / DETAIL / STOCK
- PRODUCT
  - ADD / SUB / SET / DEL
- RETURN
  - REFERER
- INSERT
  - PRINT
- RESET
- PRINT / IMAGE / FILE / VIDEO
- IF / ELIF / ELSEIF / ELSE / ENDIF

- REFRESH
- GET CART / PRODUCT

Estas opciones, permiten establecer los buffers que se emplearán en la generación de la cabecera, cuerpo y pie del carrito de la compra, así como las consultas que permitirán obtener los precios, controles de stock (limitar unidades en el carrito y actualizar el stock en el momento de la compra), descuentos y impuestos (aplicables a los productos y al total de la compra).

Notas de este módulo:

- Como se puede apreciar, este módulo usa el módulo DBAPP2 para la operación de interpretación de comandos con lo que acepta las mismas opciones que el módulo DBAPP2.
- En adelante, cuando se mencione el parámatro ROW[id] como al id del producto que contiene el carrito, también estarán disponibles para sustitución los parámetros ROW[id0], ROW[id1], ..., ROW[id{n-1}] siendo cada valor la porción de valor del id original separando por los siguientes caracteres: guión (-), guión bajo (_), punto (.), coma (,), punto y coma (;) y dos puntos (:). Esto permitirá por ejemplo, si se emplean id combinados para poder guardar productos que se encuentran en diferentes tablas (como productos y ofertas), el usar un id al llamar al comando PROCESS que sea del tipo, por ejemplo: P,id donde se dispondrá de ROW[id]="P,id", ROW[id0]="P" y ROW[id1]="id".

Comandos BEGIN y END:

Esta familia de comandos, al igual que sucede en el módulo DBAPP2, permitirán establecer los buffers que el módulo empleará para construir los resultados. El comando BEGIN y END,deben usarse conjuntamente tal como se verá en los posteriores ejemplos:

```
<!-- INCLUDE TIENDA BEGIN VOID -->
<!-- INCLUDE TIENDA END VOID -->

<!-- INCLUDE TIENDA BEGIN HEADER -->
<!-- INCLUDE TIENDA BEGIN PRODUCT -->
<!-- INCLUDE TIENDA BEGIN FOOTER -->
<!-- INCLUDE TIENDA END FOOTER -->
```

Este módulo permite definir la cabecera que usará el carrito, el detalle que se generará para cada producto y finalizar con un pie para el carrito.

Las opciones que acepta la cabecera y pie son las siguientes:

```
<!-- INCLUDE TIENDA PRINT campo -->
```

Donde el campo puede ser uno de los siguientes:

- **units**: contiene el número de unidades que se han añadido al carrito en total.
- **base**: contiene el valor de la suma de todos los totales de los productos.

- **discount**: contiene el valor de descuento que se empleará para calcular el total.
- **tax**: contiene el valor de impuestos que se empleará para calcular el total.
- **shipping**: contiene el valor de gastos de envio que se empleará para calcular el total.
- **total**: contiene el total del carrito (base – descuentos + impuestos).

Las opciones que acepta el detalle de cada producto son las siguientes:

- **price**: contiene el valor del precio de ese producto
- **units**: contiene el número de unidades que se han añadido al carrito del producto.
- **base**: contiene la base de ese producto (precio * unidades)
- **discount**: contiene el valor de descuento que se empleará para calcular el total.
- **tax**: contiene el valor de impuestos que se empleará para calcular el total.
- **shipping**: contiene el valor de gastos de envio que se empleará para calcular el total.
- **total**: contiene el total del producto (base – descuentos + impuestos).

En todas las opciones del comando PRINT, los campos price, base y total estarán siempre formateados usando la notación X.XX donde el cáracter punto (.) indica el separador decimal.

Además, en el detalle del producto, se dispondrán de todas las prestaciones de DBAPP2 (PRINT, IMAGE, FILE, VIDEO, IF, ELIF, ELSEIF, ELSE y ENDIF) si se define la consulta QUERY PRODUCT (proporciona información adicional del producto que se va a mostrar, como puede ser el nombre del producto, código de barras, imágenes, ...). Para acceder a estos datos adicionales, deberá emplearse como nombre del campo la clave ROW[campo].

Comando QUERY

El comando QUERY permite establecer las consultas que deberá hacer el sistema para obtener cierta información global del carrito y información específica de cada producto.

A continuación se listan las posibles consultas que se pueden definir:

- **PRODUCT**: permite definir una consulta que servirá para obtener la información adicional que se podrá emplear en el bloque definido entre BEGIN PRODUCT y BEGIN FOOTER. Estos datos estarán disponibles usando la clave ROW[campo]. Esta consulta dispondrá de una variable llamada ROW[id] que servirá para poder realizar el filtrado de producto. Debe explicarse que ROW[id], será un valor de cadena (no necesariamente numérico) que podrá ser la combinación de letras y números para poder ser empleado en la personalización de la consulta.

- **PRICE**: permite definir la consulta que obtendrá el precio del producto. Esta consulta podrá emplear dos variables**: ROW[id] y ROW[units] (esta última podría permitir implementar funcionalidades como precios especiales a partir de ciertas unidades). Esta consulta debe de retornar una columna con el nombre "value".

- **UNITS**: permite definir un control de unidades si se desea trabajar con stock. Esta consulta podrá emplear dos variables**: ROW[id] y ROW[units] (esta última podría permitir implementar funcionalidades como limitar las unidades de un producto para que no supere el stock actual). Esta consulta debe de retornar una columna con el nombre "value".

- **DISCOUNT**: permite definir la consulta que obtendrá el descuento del producto o del carrito en general. Esta consulta podrá emplear dos variables**: ROW[id] y ROW[units] (esta última podría permitir implementar funcionalidades como descuentos especiales a partir de ciertas unidades, o definir descuentos en ciertos productos o definir un descuento para todo el carrito en global). Esta consulta debe de retornar una columna con el nombre "value" y se interpretará como % si finaliza con este símbolo. En caso contrario, será empleado como valor absoluto que se aplicará en el ámbito que aplique.

- **TAX**: permite definir la consulta que obtendrá el impuesto del producto o del carrito en general. Esta consulta podrá emplear dos variables**: ROW[id] y ROW[units]. Esta consulta debe de retornar una columna con el nombre "value" y se interpretará como % si finaliza con este símbolo. En caso contrario, será empleado como valor absoluto que se aplicará en el ámbito que aplique.

- **SHIPPING**: permite definir la consulta que obtendrá el valor de los gastos de envio de los producto o del carrito en general. Esta consulta podrá emplear dos variables**: ROW[id] y ROW[units]. Esta consulta debe de retornar una columna con el nombre "value" y se interpretará como % si finaliza con este símbolo. En caso contrario, será empleado como valor absoluto que se aplicará al total en el ámbito que aplique.

- **MAIN**: permite definir la consulta de tipo INSERT que se ejecutará al llamar al comando INSERT (se usará este comando para finalizar la transacción y posteriormente, borrar el carrito). Esta consulta dispondrá de la mismas variables que las disponibles en el bloque entre BEGIN HEADER y BEGIN PRODUCT o BEGIN FOOTER y END FOOTER.

- **DETAIL**: permite definir la consulta de tipo INSERT que se ejecutara al llamar al comando INSERT. Esta consulta se ejecutará para cada producto que exista en el carrito y dispondrá de las mismas variables que las disponibles en el bloque entre BEGIN_PRODUCT y BEGIN FOOTER.

- **STOCK**: idem que la anterior, aunque únicamente dispondrá de las variables que no requieran de clave especial ROW[x].

Notas del comando QUERY:

- Las consultas DISCOUNT, TAX y SHIPPING, permiten definir el valor de descuentos, impuestos y gastos de envío que se emplearán en los productos y en el carrito en general. Para aplicar cada tipo de valor, deberá emplearse el parámetro ROW[id] para determinar si se trata de una consulta para un producto en concreto o si se trata de una consulta para obtener los valores que aplicarán a todo el carrito (este último caso se detecta al cumplirse la siguiente igualdad de 'ROW[id]'='').
- En el caso de que las consultas anteriores retornen un valor terminado con el símbolo "%", se calculará la parte proporcional usando como valor principal la base del producto o carrito en general.
- En el caso de que las consultas anteriores retornen un valor absoluto (no terminado con el valor "%"), el valor retornado será incrementado en el total de cada producto o carrito en general, sin aplicarse ningún cálculo adicional (es decir, que no se multiplicará por el número de productos como se podría pensar). Si se desea multiplicar el valor retornado por el número del producto, debe hacerse en la propia consulta usando ROW[units] como valor para obtener las unidades totales del producto o carrito en general.

Comando PRODUCT

Este comando permite actuar sobre el carrito y usa la siguiente sintaxis:

```
<!-- INCLUDE TIENDA PRODUCT ID ACTION NUMBER -->
```

Donde los parámetros son:

- **ID**: id del producto, que puede ser una constante de texto o un valor de sustitución permitido por DBAPP2 (ARGV[], SESSION[], CONFIG[], GET[], POST[]).
- **ACTION**: deberá de contener la acción a realizar (ADD, SUB, SET y DEL). Este parámetro acepta un valor de sustitución permitido por DBAPP2.
- **NUMBER**: deberá contener las unidades que desean aplicarse a la acción anterior (excepto en caso de la acción DEL). Este parámetro acepta un valor de sustitución permitido por DBAPP2.

El hecho de que permita usar variables de sustitución, dará gran flexibilidad de integración (tal como se verá en el ejemplo final).

Comando RETURN

Este comando envía un pseudo-código que permitirá a la aplicación volver al contenido que se desee. Si se usa la palabra clave REFERER, entonces en lugar de usar la url que se especifique como argumento de RETURN, se usará la variable SERVER[HTTP_REFERER].

Notas de este comando:

- Este módulo permite la integración en aplicaciones que usen AJAX para las tareas de recarga de contenidos. Para ello, tras la ejecución del comando RETURN, el sistema enviará el pseudo-código que permitirá capturar el evento del contenido a cargar definiendo la siguiente función:

```
<script type="text/javascript">
function _tienda_new_location(param) {
alert(param);
}
</script>
```

  El parámetro que recibe la función es la url de retorno que se empleará. En caso de no estar definida esta función, el sistema ejecutará el siguiente pseudo-código:

```
<script type="text/javascript">
window.location.href=param;
</script>
```

Comando INSERT:

Este comando, permite ejecutar las consultas de tipo INSERT definidas con QUERY MAIN, QUERY DETAIL y QUERY STOCK para completar la transacción de venta y poder finalizar el proceso.

Este comando puede recibir un parámetro adicional llamado PRINT, el cual fuerza que en lugar de ejecutarse la consulta que se generá, se muestre por la salida con la finalidad de depurar.

El detalle de como deben realizarse estas consultas esta explicado en el comando QUERY.

Comando RESET:

Este comando permite borrar el contenido del carrito de forma que pueda iniciarse una nueva transacción de venta sin productos y con el carrito vacio.

Este comando debe emplearse en el momento en que se haya validado la venta y se haya ejecutado el comando INSERT (que generará la creación de la nueva venta con el detalle de los productos y la actualización del stock).

Comando REFRESH:

Este comando permite actualizar los valores que se obtienen de las consultas PRODUCT, PRICE, UNITS, DISCOUNT y TAX, así como los cálculos derivados de las mismas.

Comando GET CART / PRODUCT:

Este comando permite acceder al carrito a bajo nivel desde un fragmento de programación PHP de forma que se puede obtener un array con el contenido del carrito o con cada producto (dependiendo del comando que se emplee).

A continuación se muestra un fragmento de código PHP que accederá al carrito desde PHP:

```
echo_buffer("<pre>");
echo_buffer(print_r(tienda("GET CART"),true));
while($prod=tienda("GET PRODUCT")) {
echo_buffer(print_r($prod,true));
}
echo_buffer("</pre>");
```

Este comando será útil cuando se desee manipular el carrito a bajo nivel o para realizar cálculos que de otro modo no se podrían realizar, tal como el cálculo en algunos casos de los gastos de envío, donde el importe de los mismos debe obtenerse usando los pesos, por ejemplo, de los productos del carrito.

Ejemplo de uso:

A continuación se mostrará un ejemplo de aplicación de tienda online donde se mostrará una tabla con productos y opciones para añadir o quitar productos del carrito. Luego se mostrará el carrito con los datos que pueden emplearse en las zonas de HEADER, PRODUCT y FOOTER, así como el caso VOID.

```
Contenido del fichero htm/test-tienda-product.htm (ubicado en templates/htm):
<!-- INCLUDE TIENDA QUERY PRICE SELECT importe value FROM tbl_productos WHERE id='ROW[id1]' -->
<!-- INCLUDE TIENDA QUERY DISCOUNT SELECT '0%' value -->
<!-- INCLUDE TIENDA QUERY TAX SELECT CASE WHEN 'ROW[id]'='' THEN '16%' ELSE '0%' END value -->
<!-- INCLUDE TIENDA PRODUCT GET[id] GET[action] GET[units] -->
<!-- INCLUDE TIENDA RETURN REFERER -->

Contenido del fichero test-tienda.htm (ubicado en templates):
<!-- INCLUDE DBAPP2 SET QUERY SELECT id,codigo,titulo,imagen,imagen_file,importe FROM tbl_productos -->
<!-- INCLUDE DBAPP2 SET LIMIT 6 -->
<!-- INCLUDE DBAPP2 SET OFFSET 0 -->
<!-- INCLUDE DBAPP2 SET NUM_TD 3 -->
<!-- INCLUDE DBAPP2 INIT -->

<table border="1" width="100%" >
<!-- INCLUDE DBAPP2 BEGIN TR_DATA -->
<tr>
<!-- INCLUDE DBAPP2 BEGIN TD_DATA -->
<td style="padding:5px;vertical-align:top" width="33%">
<!-- INCLUDE DBAPP2 PRINT titulo PREVIEW 25 --> (<!-- INCLUDE DBAPP2 PRINT codigo -->)<br/>
<div style="text-align:center">
<!-- INCLUDE DBAPP2 IMAGE imagen WIDTH 50 HEIGHT 50 FAR -->
</div>
<!-- INCLUDE DBAPP2 PRINT importe --> (
<a href="htm/test-tienda-product.htm?id=P,<!-- INCLUDE DBAPP2 PRINT id -->&action=ADD&units=1">ADD</a>
<a href="htm/test-tienda-product.htm?id=P,<!-- INCLUDE DBAPP2 PRINT id -->&action=SUB&units=1">SUB</a>)
</td>
<!-- INCLUDE DBAPP2 END TD_DATA -->
</tr>
<!-- INCLUDE DBAPP2 END TR_DATA -->
</table>

<br/>

<!-- INCLUDE TIENDA BEGIN VOID -->
<table border="1" width="100%">
<tr>
<td style="padding:5px;vertical-align:top">
CARRITO VACIO
</td>
</tr>
</table>
<!-- INCLUDE TIENDA END VOID -->

<!-- INCLUDE TIENDA QUERY PRODUCT SELECT codigo,titulo,imagen,imagen_file FROM tbl_productos WHERE id='ROW[id1]' -->
<table border="1" width="100%">
<!-- INCLUDE TIENDA BEGIN HEADER -->
<tr>
<td style="padding:5px;vertical-align:top">
Header:<br/>
Units: <!-- INCLUDE TIENDA PRINT units --><br/>
Base: <!-- INCLUDE TIENDA PRINT base -->&euro;<br/>
Discount: <!-- INCLUDE TIENDA PRINT discount --><br/>
Tax: <!-- INCLUDE TIENDA PRINT tax --><br/>
```

```
Shipping: <!-- INCLUDE TIENDA PRINT shipping --><br/>
Total: <!-- INCLUDE TIENDA PRINT total -->&euro;
</td>
</tr>
<!-- INCLUDE TIENDA BEGIN PRODUCT -->
<tr>
<td style="padding:5px;vertical-align:top">
Product:<br/>
Id: <!-- INCLUDE TIENDA PRINT id --><br/>
Price: <!-- INCLUDE TIENDA PRINT price -->&euro;<br/>
Units: <!-- INCLUDE TIENDA PRINT units --><br/>
Base: <!-- INCLUDE TIENDA PRINT base -->&euro;<br/>
Discount: <!-- INCLUDE TIENDA PRINT discount --><br/>
Tax: <!-- INCLUDE TIENDA PRINT tax --><br/>
Shipping: <!-- INCLUDE TIENDA PRINT shipping --><br/>
Total: <!-- INCLUDE TIENDA PRINT total -->&euro;<br/>
Extra:<br/>
Titulo: <!-- INCLUDE TIENDA PRINT ROW[titulo] --><br/>
Codigo: <!-- INCLUDE TIENDA PRINT ROW[codigo] --><br/>
Imagen: <!-- INCLUDE TIENDA IMAGE ROW[imagen] WIDTH 50 HEIGHT 50 FAR -->
</td>
</tr>
<!-- INCLUDE TIENDA BEGIN FOOTER -->
<tr>
<td style="padding:5px;vertical-align:top">
Footer:<br/>
Units: <!-- INCLUDE TIENDA PRINT units --><br/>
Base: <!-- INCLUDE TIENDA PRINT base -->&euro;<br/>
Discount: <!-- INCLUDE TIENDA PRINT discount --><br/>
Tax: <!-- INCLUDE TIENDA PRINT tax --><br/>
Shipping: <!-- INCLUDE TIENDA PRINT shipping --><br/>
Total: <!-- INCLUDE TIENDA PRINT total -->&euro;
</td>
</tr>
<!-- INCLUDE TIENDA END FOOTER -->
</table>

<br/>

<table border="1" width="100%">
<tr>
<td style="padding:5px;vertical-align:top">
<!-- INCLUDE TIENDA QUERY MAIN INSERT INTO tbl_pedidos(id,unidades,base,descuento,iva,total)
VALUES(NULL,'ROW[units]','ROW[base]','ROW[discount]','ROW[tax]','ROW[total]') -->
<!-- INCLUDE TIENDA QUERY DETAIL INSERT INTO
tbl_pedidos(id,id_producto,codigo,precio,unidades,base,descuento,iva,total)
VALUES(NULL,'ROW[id1]','ROW[codigo]','ROW[price]','ROW[units]',
'ROW[base]','ROW[discount]','ROW[tax]','ROW[total]') -->
<!-- INCLUDE TIENDA QUERY STOCK UPDATE tbl_productos
SET stock=stock-ROW[units] WHERE id='ROW[id1]' -->
<!-- INCLUDE TIENDA INSERT -->
</td>
</tr>
</table>
```