xaWeb

Desarrollo web rápido y fácil

Por Ignacio Ortiz de Zúñiga

Guía de programación

xaWeb es una nueva plataforma de desarrollo creada por OZ Software que permite realizar aplicaciones de gestión para entornos Web y sistemas operativos Windows (32 bits) y Linux (64 bits). xaWeb se basa en la tecnología CGI para la creación de páginas web de contenido dinámico. La tecnología CGI tiene importantes ventajas sobre otros entornes de desarrollo basados en guiones como PHP, como son:

- Velocidad, ya que se trata de código compilado
- Seguridad, ya que se impide la ejecución de guiones que un 'hacker' haya podido subir a nuestro servidor
- Salvaguarda de los derechos de propiedad intelectual del desarrollador al no tener que alojar en el servidor web ningún código fuente, únicamente el ejecutable.

Este documento no pretender ofrecer una información completa de todo lo que es y hace xaWeb, ni tampoco explicar el funcionamiento de la tecnología CGI. Se pretende únicamente explicar los fundamentos de la herramienta y a grandes rasgos su sistema de funcionamiento para que cualquier programador entienda lo que realmente está sucediendo por debajo. Entender cómo funciona el software ayuda muchísimo a hacer buen software y a resolver los problemas diarios de programación de la forma correcta. **Es importante que se lea este documento antes de empezar a trabajar con xaWeb. El proceso de aprendizaje será mucho más rápido y con menos errores**.

OBJETIVOS

xaWeb está dirigido principalmente a aquellos desarrolladores **Harbour** que necesitan resolver importantes áreas de sus desarrollos en la web, sin que ello suponga renunciar al uso de su lenguaje preferido. El 'framework' básico que se incluye con xaWeb pretende cubrir las necesidades de cualquier desarrollador de software de gestión empresarial, sin que deba tener unos grandes conocimientos de HTML, JavaScript o CSS. Consideramos a xaWeb una herramienta claramente 'full-stack' enfocada a pequeños equipos de desarrollo y desarrolladores individuales.

El objetivo principal es que cualquier usuario de Harbour empiece a ser productivo desde el primer día con xaWeb y no vea grandes diferencias de programación entre el entorno de escritorio y xaWeb. Esto suponía un reto importante ya que la programación Web y el entorno de desarrollo de escritorio

se parecen muy poco. El primer paso fue crear una completa jerarquía de clases, pero que ocultase la complejidad del HTML y JavaScript. Existen controles específicos en xaWeb para cada uno de los controles existentes en HTML y el proceso de interpretar y producir el código HTML es realizado de forma automática por xaWeb.

Se ha hecho un gran esfuerzo para que esta jerarquía no se comporte como una camisa de fuerza que impida al desarrollador tener todo el potencial que ofrece HTML y la programación JavaScript. Absolutamente cualquier '**tag'** o evento de cualquier elemento HTML puede ser utilizado en xaWeb sin el mayor problema. No hay limitaciones. Cualquier evento, de cualquier control se puede sobrecargar en xaWeb. No obstante, el principal objetivo de xaWeb es ofrecer soluciones Web a los clásicos procesos de gestión empresarial con el mayor pragmatismo posible y simplificando al máximo los conocimientos de programación HTML, CSS y JavaScript por parte del programador.

El segundo objetivo fue la transferencia de datos automática y bidireccional entre las páginas web creadas con xaWeb y la aplicación CGI sin que el desarrollador tenga que hacer nada para que este mecanismo de comunicación funcione.

El tercer objetivo absolutamente necesario es que xaWeb pudiese hacer aplicaciones Web tanto en entorno Linux como Windows. Objetivo que también se ha conseguido.

Por último, establecimos una serie de requisitos que entendíamos son absolutamente necesarios y suponen ciertos cambios con respecto al desarrollo de aplicaciones de escritorio para el entorno Windows, que son:

- Uso de UTF-8 como único juego de caracteres: Esto puede suponer un problema para entornos de datos donde se desea acceder a los mismos desde aplicaciones de escritorio Xailer y aplicaciones xaWeb. No obstante, la solución es muy sencilla, al menos, en bases de datos MySQL o MariaDB ya que sólo tiene que poner la instrucción "*set names latin1*" una única vez cuando arranca la conexión en su aplicación de escritorio (internamente deberá crear las tablas con UTF-8).
- Evitar tener que enlazar ninguna librería de Xailer. Es decir, xaWeb es completamente independiente de Xailer y no requiere de ninguna de sus librerías.
- Las aplicaciones debían de ser de 64 bits en Linux ya que es el entorno habitual en dicho sistema operativo.
- xaWeb debería ser independiente de cualquier API de Windows, ya que estas APIs no existen en Linux.

Recomendamos utilizar el IDE de Xailer para el desarrollo de aplicaciones por varios motivos:

- Facilidad para establecer el tipo de proyecto: CGI Windows o CGI Linux
- Soporte de Intelisense muy avanzado de todas las librerías de xaWeb
- Depuración en entorno Windows

No obstante, aquellos que estén acostumbrados a utilizar la herramienta hbmk2.exe y un buen editor de texto como Visual Studio Code no tendrán ningún problema en seguir usando su editor favorito.

ARQUITECTURA

xaWeb consta de los siguientes componentes:

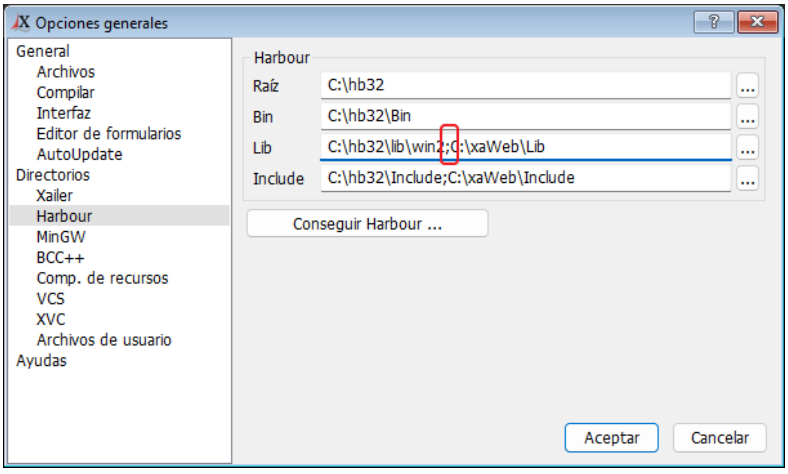
- Una librería estática con todas las fuentes de xaWeb
- Una librería adicional estática para permitir la sobrecarga de clases por el usuario
- Un fichero *include* de nombre "xaweb.ch" que deberá incluir en todos sus módulos
- Varios módulos CSS y JavaScript de muy poco peso

Todas las clases tiene un ancestro vacío que sirve para que el usuario sobrecargue la clase sin tener que heredar en una nueva clase. Con un ejemplo se entiende más fácilmente: La clase WEdit representa un control tipo Input-Text de HTML. No obstante, si se dirige a las fuentes de xaWeb observará que la clase tiene únicamente este código:

```
CLASS WEdit FROM ZEdit
ENDCLASS
```

La clase que realmente tiene todo el código es **ZEdit**, pero el usuario nunca instancia clases '**Z**', sino clases '**W**'. Con esto conseguimos que, si el usuario quiere, por ejemplo, añadir una propiedad más a la clase, no tiene más que incluir en su propio código una definición completa de '**W**' con todas sus mejoras y con los cambios que desee.

Recomendamos es instalar xaWeb en la carpeta 'c:*xaWeb'* ya que todos los ejemplos tienen indicado ese directorio para las librerías de xaWeb. No obstante, puede instalar xaWeb en la carpeta que desee siempre que configure correctamente el camino donde el IDE de Xailer ha de buscar las librerías de xaWeb. Colgando de esa carpeta encontrará las carpetas 'lib', 'include' y 'source'. En el caso de que utilice el IDE de Xailer, para que sus proyectos xaWeb pueda encontrar con facilidad el fichero include "xaWeb.ch" es recomendable que incluya dicha carpeta en los directorios de búsqueda de Harbour. Se haría de la siguiente forma:



Observe que tanto en el campo LIB, como en el campo INCLUDE se ha introducido un ';' para añadir más de un camino de búsqueda. Es necesaria la versión de Xailer 9.2 (o Xailer 9.1 Beta 3) para que esta funcionalidad esté disponible.

Cuando realice proyectos para Windows deberá indicar en el proyecto las librerías de xaWeb para Windows, que son: xaWebWin y wxaWebWin. Cuando realice proyectos para Linux deberá utilizar las librerías xaWebLinux y wxaWebLinux.

Para poder utilizar ***Materialize*** con xaWeb deberá incluir su librería de nombre "xaWebMaterializeLinux" o "xaWebMaterializeWin" dependiendo de si está realizando ejecutables para entorno Linux o Windows. Además, deberá incluir en todos los ficheros PRG el fichero include "**xa-materialize.ch**" en vez de "xaWeb.ch".

NOTA IMPORTANTE

La librería de Materialize debe de incorporarse a las librerías de enlazado en la configuración del proyecto antes de las librerías de xaWeb.

TECNOLOGÍA CGI

La plataforma de desarrollo **xaWeb** está basada en el uso de la tecnología CGI que para más información me remito a los siguientes enlaces:

- Interfaz de entrada común - Wikipedia, la enciclopedia libre
- Common Gateway Interface (CGI): Qué es y cómo funciona (godaddy.com)

La funcionalidad básica principal de cualquier aplicación CGI es devolver el contenido de una página HTML a través de la salida estándar (STDOUT). La ejecución del CGI se hace de la misma forma que se accedería a una página web estática a través de una URL, por ejemplo: http://www.ejemplo.com/index.html

Para que el servidor web ejecuta la aplicación y devuelve el contenido generado en STDOUT es necesario que éste sepa que es un archivo que ha de ejecutar y devolver el contenido de STDOUT en vez de leer y retornar el contenido del archivo. Para ello es necesario utilizar una extensión que el servidor web pueda reconocer como archivo ejecutable. En el caso de servidores web bajo Windows puede utilizar la extensión 'exe' como suele ser habitual. En Linux recomendamos utilizar la extensión 'cgi'.

Las aplicaciones realizadas por xaWeb además de comportarse como un clásico CGI que devuelve el '*stream*' HTML que mostrará el navegador, también son capaces de comportarse como un servicio

web (‘Web service’) que normalmente retorna un objeto JSON. Este tipo de operaciones es explicado en un capítulo posterior.

La ubicación de los archivos ejecutables CGI en el servidor deben de estar situados en carpetas ad-hoc para dicho contenido en las cuales sólo haya permisos de ejecución sobre los CGIs que se hayan instalado. En Apache web server el contenido de los CGI se encuentra en el directorio ***/usr/lib/cgi-bin***.

CGI COMO GENERADOR DE CÓDIGO

Es muy importante que **nunca olvide** que su código xaWeb lo único que hace es generar código HTML, código JavaScript y reglas CSS. Lo que realmente se ejecuta en su página HTML es el código generado por el CGI, pero éste ya no está presente. Es un error muy habitual intentar introducir código Harbour en propiedades de objetos que luego se van a desplegar. Un ejemplo lo hace más evidente:

```
oButton:OnClick := {|| Msginfo(“On click”)}
```

Esto no funciona. La propiedad *OnClick* del botón ha de tener un código que el navegador sea capaz de entender.

PASO DE PARÁMETROS AL CGI

El paso de parámetros a la aplicación CGI se hace normalmente a través de un método GET, que consiste en incluir en la propia URL dicha información: después de la URL propiamente dicha se incluye el carácter ‘?’ y a continuación los duplos de los parámetros separados por el carácter ‘&’ y con el formato ‘nombre’=’valor’. Como en las direcciones URL no admiten caracteres especiales y espacios hay que hacer un proceso simple de conversión de dicha cadena. Ese proceso se denomina ‘URL encode’ y lo hace automáticamente xaWeb. La conversión de tipo del elemento ‘valor’ a tipo carácter también es realizado automáticamente por xaWeb. Otro método fundamental para el paso de información desde la página web al servidor es a través del método POST que es el que se usa por ejemplo cuando se pulsa el botón ‘*submit*’ de un formulario. En dicho caso la información va contenido en un ‘*stream*’ aparte de la URL y es recibido directamente por el CGI.

EJECUCIÓN DEL CGI

El código de inicialización del CGI consistiría básicamente en instanciar un objeto de nuestra clase heredada de ***WDoc*** que incluye todo el código HTML que queremos devolver y punto, pero obviamente esto no es posible realizarlo de forma directa, es necesario realizar una serie de operaciones para que nuestra aplicación se pueda comunicar con el servidor web. La clase que se encarga de empezar todo el proceso es la clase ***WApplication*** y esta es instanciada de forma automática a través de un INIT PROCEDURE. Dicho procedimiento instancia un objeto (singleton) de la clase ***WEngine*** que es la única que realmente se comunica con el servidor web.

En el código de nuestra función 'MAIN' indicaremos el documento que deseamos procesar. En xaWeb, ese documento se considera el documento por **defecto**, pero a través del paso de parámetros es posible cambiarlo por otro cualquiera. Este sería el código clásico de nuestra PROCEDURE Main:

```
PROCEDURE Main()
   Application:cTitle := "Hello World"
   WRouter():New( Application ):Start( "WDocMain" )
   Application:Run()
RETURN
```

En este ejemplo la clase que se va a intentar instanciar un objeto es ***WDocMain*** que necesariamente ha de heredar de la clase ***WDoc***. Para conseguir que nuestro CGI sea capaz de instanciar un documento distinto, o incluso ejecutar un método en concreto de uno de los documentos se utiliza la clase ***WRouter*** cuya labor es básicamente esa: decidir qué operación se debe de realizar, con que documento, él cual se encarga de instanciar y si ha de ejecutar algún método en concreto de dicho documento.

Cuando la URL no incluye ningún parámetro, se instancia el documento indicado por defecto y se ejecuta su método ***CreateDoc.*** Dicho método es el encargado de instanciar todos los elementos que mostrará la página web.

La clase Engine

Algunos de los conceptos que se indican en este capítulo requieren de conocimientos básicos del funcionamiento del protocolo HTTP.

El objeto ***WApp*** instancia un '*singleton*' de la clase ***WEngine*** que se almacena en la variable pública '***Engine'***. Dicho objeto contiene información importante sobre la comunicación con el servidor web:

- Cabecera (Header) enviada desde la página web que se guarda en la propiedad *hResHeaders*
- Variables de entorno clásicas del modelo CGI instanciadas por el servidor web con información de la petición que se guarda en la propiedad *hEnvironment.* Como, por ejemplo*:* AUTH_TYPE, ANNOTATION_SERVER, CONTENT_TYPE, CONTENT_LENGTH, DOCUMENT_ROOT.
- Las cookies recibidas son también almacenadas en el hash *hReqCookies*
- Los parámetros recibidos vía GET son almacenados en el hash *hParams*
- *cPostBuffer* almacena el *stream* recibido vía POST. Existe igualmente un hash *hPost* que tiene información para el que el caso en el que *stream* tenga información de formularios. En cuyo caso el hash tiene los valores de los campos del formulario.

El objeto **Engine** guarda más información de muchísima utilidad que se explicará en un capítulo posterior.

La clase *WDoc* es la responsable del envío de toda la información a *WEngine*, operación que se realiza de forma automática cuando se abandona la aplicación CGI. xaWeb controla de forma automática

todo el flujo de información hacia STDOUT. El programador NUNCA debe de enviar un dato directamente a STDOUT, ni siquiera a través del objeto **Engine**.

SESIONES Y COOKIES

En programación web es necesario poder reconocer desde el servidor web al usuario que realiza más de una petición URL en un breve espacio de tiempo. Las cookies nos ofrecen esta funcionalidad de una forma muy sencilla. Básicamente una cookie es un archivo que se crea en la máquina cliente, pero que es accesible también desde el servidor web. xaWeb utiliza las cookies para sincronizar la información existente en la página web y la aplicación CGI como veremos más adelante.

Una cookie especial en xaWeb, es la cookie de sesión, que es un simple ID que mantiene su valor inalterable hasta que caduca por llevar un tiempo sin llamarse a la URL del CGI. Dicho tiempo de caducidad se establece en la propiedad *WEngine:nSessionTTL.* Desde el CGI podemos acceder a su valor a través del método *WEngine:SessionId.*

El gestor interno de mantenimiento de sesiones por defecto se recupera internamente llamando a la función ***Session***() que retorna un "singleton" del gestor, que por defecto, es una instancia de la clase **WSession**. Este objeto es accesible mediante la propiedad *Document:oSession*. Este gestor que incorpora xaWeb, será para muchos usuarios suficiente, pero usted puede realizar su propio gestor y crear la función singleton **Session**() que retorne una instancia de su clase personal. Esta clase se encarga de grabar y recuperar un *hash* que le suministra la clase **WDoc** en un fichero JSON dentro de la carpeta '*sessions'*. Analizando el código de *WSession*, le será muy sencillo, sobrecargar dicha clase para que los métodos **Save** y **Load** trabajen por ejemplo con una base de datos en vez de con ficheros planos.

Si utiliza servidores Linux para alojar sus páginas es necesario que cree la carpeta '**sessions'** a partir de la raíz **'usr/lib/cgi-bin**' y otorgue los privilegios de escritura de **grupo** al usuario 'www-data' (Apache). Para ello deberá utilizar las herramientas de Linux **chown** y **chmod** de la siguiente forma:

```
sudo chown :www-data /usr/lib/cgi-bin/sessions
sudo chmod g+rw  /usr/lib/cgi-bin/sessions
```

La carpeta base donde se ubican las sesiones es en HB_DirBase() + "sessions". A partir de esta carpeta, se crea una carpeta adicional con el mismo nombre que el ID de sesión y dentro de esa carpeta se encuentro el archivo tipo JSON de nombre "session.json" que es el que guarda toda la persistencia de la sesión.

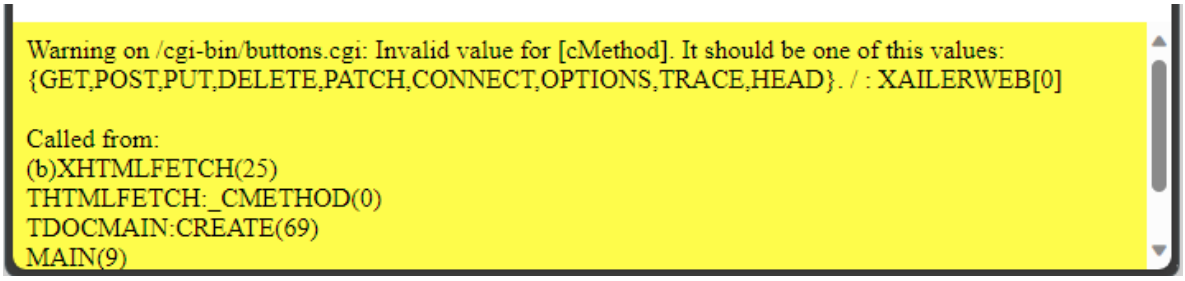
El gestor de sesiones de xaWeb le permite salvar en la carpeta de sesión todo lo que desee. Como ha podido leer, se crea una carpeta por cada sesión y, por lo tanto, desde su CGI, podrá grabar en dicha carpeta cualquier archivo adicional que necesite. Lógicamente todo lo que guarde en esa carpeta se pierde cuando caduque la sesión. La carpeta de la sesión activa es fácilmente accesible con la propiedad *Document:oSession:SessionPath().*

MODO DEPURACIÓN

El objeto *Engine* incorpora una propiedad de nombre *lDebug* que cuando se pone a verdadero nos proporciona las siguientes funcionalidades:

- Genera un archivo con toda la información que se ha enviado al servidor web, incluidas las cabeceras y cookies. El fichero es por defecto 'error.log' en el mismo directorio que el CGI, pero se puede modificar a través de la propiedad *WEngine:cLogFile*.
- Indica el tiempo de ejecución del CGI en el archivo error.log.
- Muestra avisos de uso inadecuado de propiedades y métodos de los distintos controles HTML.

Además de este modo de depuración xaWeb incorpora una función de nombre *LogDebug(xVal)* que directamente muestra en el propio navegador (iFrame) cualquier contenido que deseemos.



Es posible depurar paso a paso con el IDE de Xailer únicamente realizando ejecutables para Windows, y éste es el único caso donde deberá enlazar con la librería Xailer.lib. En el resto de los casos y para el ejecutable final, no es necesario enlazar con dicha librería e incluso se recomienda no hacerlo.

NOTA IMPORTANTE

Cuando enlace con la librería Xailer.lib para poder depurar paso a paso, es imprescindible que evite que la librería Xailer.lib se enlace en modo "Revisar en primer lugar" que es su estado por defecto y se puede comprobar por estar el nombre la librería en negrita. Para cambiar a modo normal utilice el menú contextual del propio elemento 'Xailer.lib'. La librería Xailer.lib debe enlazarse después de las librerías de xaWeb y xaMaterialize.

ESTRUCTURA DE DIRECTORIOS EN LA WEB

Es importante tener claro como los servidores Web sirven las páginas y donde esperan encontrar los distintos archivos que pretendemos cargar. Lo más sencillo para saber dónde el navegador pretende

encontrar cualquier fichero es utilizando la variable de entorno CGI '*DOCUMENT_ROOT*', que se puede acceder fácilmente con la instrucción:

```
Engine:DocumentRoot()
```

Este es el directorio donde el servidor Web (Apache) prevé encontrar todos sus ficheros. Pero este directorio, no es el mismo en el cual se apoyará nuestro CGI para encontrar otros ficheros. Como norma básica:

- Utilice Engine:DocumentRoot() cuando se trate de archivos que tiene que acceder el servidor web, como por ejemplo, archivos Html, Jpg, Png, Css, Js. Que suele ser: '/var/www/html' (Linux)
- Utilice HB_DirBase() cuando se trate de archivos que tiene que acceder **internamente** desde el CGI

Cuando queramos que nuestras páginas web accedan a un recurso en el servidor web sólo habrá que poner el 'path' a partir de Engine:DocumentRoot(). Por ejemplo: "/css/style.css" (realmente estaría en /var/www/html/css/style.css). Ahora bien, si deseamos crear o guardar un fichero en esa carpeta desde nuestro CGI habrá que indicar el camino completo.

Cuando queramos que nuestro CGI acceda a ficheros externos para lectura o escritura deberemos utilizar siempre HB_DirBase() para indicar el camino. Si no utilizamos HB_DirBase() para indicar el camino, estaremos indicando un 'path' relativo desde la ubicación del propio CGI.

Es importante recalcar la importancia de poner o no la barra ("/") al principio de cualquier 'path' relativo. Si pone la barra está indicando el directorio raíz de Engine:DocumentRoot(). Es decir, donde se encuentran los archivos HTML (por ejemplo). Si no pone la barra está indicando un 'path' relativo a la ubicación del CGI.

En los servidores Web Linux, normalmente el CGI se ubica en **`'/usr/lib/cgi-bin'`**, mientras que las páginas web se ubican en **`'/var/www/html'`**. Por lo tanto, *Engine:DocumentRoot* apunta a este último directorio.

No obstante, cualquier archivo que intente cargar **DIRECTAMENTE** desde su propio CGI será relativo a la ubicación del CGI y no a *Engine:DocumentRoot.* Si no incluye ningún '*path*' el archivo debería encontrarse en el mismo directorio que el CGI. Si crea una subcarpeta en el directorio '*cgi-bin*' y quiere acceder a un archivo que se encuentre en dicha carpeta la recomendamos que utilice la función de Harbour `HB_DirBase() + '/carpeta'.`

No es lo mismo que el CGI acceda **directamente** a un archivo, a que las páginas web que genera el CGI quieren acceder a un archivo. La diferencia es total y es importante que entienda la diferencia y más aún con servidores Linux. De ahí la importancia que se ha pretendido indicar poniendo en negrita la palabra 'directamente'.

Este problema no ocurre en programación tradicional HTML (sin CGIs), ya que el fichero de carga inicial de la página web que suele llamarse *'index.html'* se encuentra en `'/var/www/html'` y por lo tanto los *'paths'* relativos coinciden con los *'paths'* absolutos.

NOTA IMPORTANTE

No es una buena idea el que nuestro CGI genere archivos que luego van a ser abiertos por el servidor Web, como hojas de estilo, código JavaScript o imágenes, ya que es muy posible que otra instancia de ese mismo CGI modifique esos archivos que acaba de crear. Y si siempre son los mismos archivos, tiene más sentido incluirlos en la página web como una referencia externa.

EJECUCIÓN DE CGIS EN LINUX

En entornos Linux va a ser necesario que establezca los permisos necesarios para que los CGIs se ejecuten correctamente. Recuerde que la ubicación de éstos **nunca** es donde se ubican los archivos Html, css, js e imágenes. En servidores Apache, bajo instalaciones estándar, la ubicación se encuentra en la carpeta */usr/lib/cgi-bin*, pero si usted tiene contratado un servidor Linux con software de gestión de panel, como *Plesk* o *Cpanel*, es muy probable que tenga que ubicar los archivos CGI en una carpeta diferente e incluso tenga que realizar alguna configuración en Apache Web server. Debe de consultar la documentación de su panel para saber exactamente dónde debe subir los archivos CGI para que estos se ejecuten sin problemas. Incluso aunque suba los archivos a la carpeta adecuada e indique correctamente el camino de búsqueda en el navegador, es probable que no le funcione hasta que no establezca correctamente los privilegios del archivo, que han de incluir la opción de '**ejecución'**. El primer paso es establecer al usuario 'www-data' como **grupo** con permisos en la carpeta /usr/lib/cgi-bin con el siguiente comando:

```
sudo chown :www-data /usr/lib/cgi-bin
```

El comando *sudo* solicita permisos de administrador y el comando *chown* es el que realmente establece los permisos. El siguiente paso es establecer los permisos de lectura y ejecución en dicha carpeta:

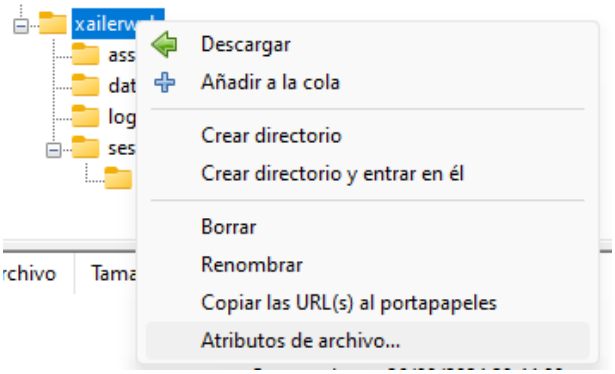
```
sudo chmod g+rx /usr/lib/cgi-bin
```

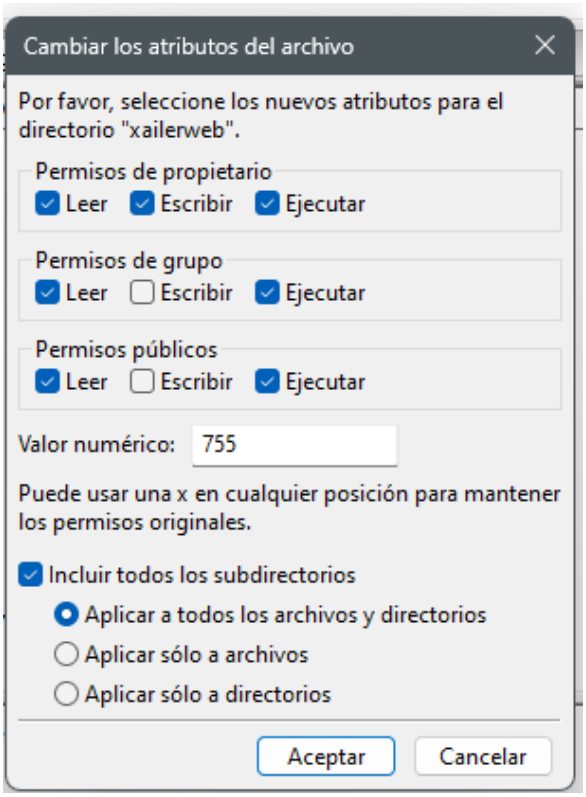
Y a continuación deberá establecer permisos en cada archivo CGI:

```
sudo chmod 755 /usr/lib/cgi-bin/myproject.cgi
```

Si cuando intenta ejecutar el CGI desde el navegador recibe un error 500, con mucha probabilidad se tratará de un error de falta de permisos. Si el navegador no es capaz de encontrar su CGI recibirá un error 404.

Si utiliza una herramienta como Filezilla para subir los archivos, toda esta configuración se simplifica completamente utilizando la opción de menú contextual 'Atributos de archivo' que tiene el software:





Tendrá que establecer los permisos (755) en cada nuevo archivo CGI que suba a su servidor Linux, ya que los nuevos archivos que suba no tendrán por defecto esos valores. No obstante, esta operación sólo tendrá que realizar una única vez. Las próximas subidas que realice del CGI mantendrán los permisos que tuviese el archivo que sustituye. Para solucionar este problema existe una utilidad de nombre 'setfacl' que permite establecer los permisos por defecto para nuevos archivos. Recomendamos el siguiente enlace para más información.

Si aún no consigue ejecutar sus CGIs, el siguiente paso sería examinar el archivo 'suexec.log' del servidor web que en Ubuntu se encuentra en /var/log/apache2/suexec.log. Necesitará acceder mediante SSH a su servidor con la utilidad Putty o similar.

No se deje impresionar por la complejidad de todo lo explicado. Es más fácil de lo que parece.

Los Eventos

Posiblemente ya sepa lo que es un evento y esté acostumbrado a usarlos. No obstante, eventos como tales, no existen en *Harbour* y hay que recurrir al uso de *code-blocks* para conseguir una funcionalidad

similar. Un evento es una operación realizada por parte del usuario final que puede ser controlada por parte de nuestra aplicación. Un ejemplo clásico es la pulsación de un botón.

En xaWeb hay que distinguir entre dos tipos de eventos:

1. Los que se disparan desde nuestro CGI y son procesador por el CGI
2. Los que se disparan desde el navegador, pero que son procesados por nuestro CGI o en el propio navegador

El primer tipo de eventos son los clásicos ya existentes en Xailer que muchos conoceréis. Este el caso de los eventos:

- oControl:OnPreProcess()
- oControl:OnDeploy(@cHtml)
- oDoc:OnDeployHead(@cHaead)
- oDoc:OnDeployBody(@cBody)

Estos eventos se les puede asignar el nombre de un método del documento principal (al igual que en Xailer) o un bloque de código. En ambos casos, el primer parámetro es siempre el control o documento que dispara el evento. Lo sucesivos parámetros dependerá del tipo de evento.

```
Document:OnDeployBody := “MiMetodo”
Document:OnDeployBody := {|oDoc, cHtml| … }
```

El segundo tipo de eventos se parece también al existente en Xailer, pero con mayor funcionalidad e importantes diferencias:

1. Ejecución de un método de clase: Si indicamos un literal (oBtn:OnClick := “miMetodo”) xaWeb buscará un método con ese nombre en el documento activo. Si lo encuentra, hará que al pulsar el botón se recargue el CGI indicando en sus parámetros que tiene que ejecutar el método “miMetodo”. El método recibe un parámetro ***hEvent*** que es un *hash* con el ID del control que ha disparado el evento, las coordenadas X e Y de pulsación, y si las teclas Control y Alt estaban pulsadas al hacerse clic.

2. Ejecución de una función JavaScript: Si como en el caso anterior, indicamos un literal, pero xaWeb no encuentra ningún método de clase con ese nombre, hará que la pulsación del botón provoque la llamada a una función JavaScript con ese nombre.

```
TEXT INTO cJs
  function myfunction(e) {
    alert('Click triggered from a user function');
  }
ENDTEXT

WITH OBJECT WButton():New( Document )
  :cText := "This button fires a Javascript user function"
  :OnClick := "myfunction"
```

```
    :cId := "button2"
END WITH

Document:AddScript( cJs )
```

3. Ejecución de código Javascript: Si introducimos un literal entre etiquetas <script> y </script>, al pulsar el botón desde la página web se ejecutará el código que hayamos introducido. Por ejemplo: `<script>alert("OnClick")</script>`.

4. Asignación de un objeto: Si asignamos un **objeto** al evento, éste ha de tener forzosamente un método de nombre ***Html*** que es el que se ejecutará cuando se realice el despliegue y el valor devuelto por el método será el que se asigne en el HTML desplegado.

> **NOTA IMPORTANTE**
>
> En los dos primeros casos, para que los eventos funcionen correctamente, es necesario que los controles HTML tengan asignada su propiedad cID.

El Evento OnValidate

Este evento es un evento especial que se produce cada vez que un control tipo <**Input**> cambia de valor y cuando se realiza el '**submit'** de formulario. Este evento será tratado con mayor detenimiento en el apartado de formularios, pero es importante que lo tenga en consideración desde el principio. Básicamente las diferencias con otros eventos estándar, es que han de devolver un objeto JSON con información de si procede validar o no el valor introducido en el control y en el caso de que no, indicar un mensaje de error.

El Evento OnFormShow

Este evento es un evento especial que se produce cada vez que se muestra un formulario. Este evento será tratado con mayor detenimiento en el apartado de formularios, pero es importante que lo tenga en consideración desde el principio. Básicamente las diferencias con otros eventos estándar, es que han de devolver un objeto JSON con los valores de inicialización de cada control de entrada de datos.

La clase Doc y DocSection

Como ya se ha comentado en un apartado anterior, la clase *WDoc* es la responsable de hacer el despliegue del contenido HTML. Una aplicación xaWeb puede tener múltiples objetos *WDoc*, pero sólo uno de ellos puede ejecutarse cada vez que se ejecuta el CGI. A través del paso de parámetros

podemos especificar exactamente el objeto *WDoc* que deseamos instanciar y, por tanto, éste será el responsable de desplegar el contenido HTML. Una vez que un objeto *WDoc* ha sido instanciado, éste es accesible a través de la variable pública '**Document'**.

El objeto *WDoc* puede ser el responsable del despliegue de todo el contenido HTML, pero lo conveniente es utilizar para separar cada parte del contenido. Por ejemplo: la cabecera podría ser una sección, el pie de página otro y el cuerpo, otro más. Las secciones en xaWeb se crean con el método *Document:AddSection(<cName>)* y retornan un objeto del tipo *WDocSection.* Incluso cuando no haya creado ninguna sección, xaWeb crea de forma automática una sección inicial por defecto con el nombre "*default*".

Un objeto *WDoc* puede tener múltiples secciones u objetos *WDocSection*. Incluso dos WDoc distintos pueden compartir varias secciones, como sería el caso de compartir un pie de página. El hecho de crear una sección no significa que tenga que desplegarse forzosamente. El objeto *WDocSection* tiene una propiedad de nombre ***lDeploy*** que cuando está a verdadero indica que la sección debe ser desplegada.

Todas las secciones son desplegadas en HTML dentro de un contenedor HTML del tipo <x-doc-section></x-doc-section>. El nombre indicado en el parámetro de AddSection() se utiliza como ID en dicho elemento HTML. Otra de las propiedades importantes del objeto WDocSection es ***lHide***. Esta propiedad permite ocultar la sección, pero desplegándola igualmente: simplemente pone el estilo '*display:none'* al elemento HTML. La utilidad de esta propiedad es la de poder crear secciones ocultas que por código JavaScript podemos hacer visibles a nuestro antojo, como por ejemplo un formulario.

El orden de creación de las secciones, marca en principio el orden de despliegue de estas en el HTML. No obstante, se puede cambiar utilizando la propiedad ***lFooter*** que obliga a que la sección vaya después del cuerpo (body) principal HTML.

Cuando se ejecuta el CGI sin ningún parámetro, sólo se instancia el objeto *WDoc* por defecto y se realiza una llamada a su método *CreateDoc*. Después de la llamada a *CreateDoc*, el documento se despliega y se sale de la aplicación. No obstante, a través de parámetros GET, podemos indicarle un documento distinto o un método en concreto a ejecutar. Por ejemplo:

http://localhost/test.cgi?**action**=mydoc-mymethod

Esta URL solicita la carga del documento '*mydoc'* y la ejecución de su método '*mymethod'* después de que éste haya sido instanciado y se haya ejecutado su método *CreateDoc*.

Cuando necesitamos que una o más secciones sean instanciadas después de la llamada a '*mymethod'* y no antes, podemos utilizar el método ***RegisterSection(<cName>)*** de la clase *WDoc* (*lDeploy* a de estar a falso), que recibe como único parámetro el nombre otorgado a la sección. La ventaja de utilizar este método es que las secciones registradas son instanciadas después del método elegido en 'action' y por lo tanto pueden acceder a datos ya actualizados.

SINCRONIZACIÓN DE ENTORNOS WEB Y CGI

xaWeb incorpora en todos sus ejecutables un código JavaScript determinístico (que es siempre el mismo) que puede ir incrustado en la aplicación o en un módulo JavaScript aparte (xw_backpack.js). Dicho módulo contiene unas cuantas funciones que utilizará xaWeb de forma interna sin que el programador tenga que conocer su funcionamiento. Además de este código determinístico, xaWeb incluye un *'script'* adicional no determinístico que varía en base al código xaWeb que se haya ejecutado. Dicho 'script' realiza básicamente las siguientes operaciones:

- Crear variables para indicar los elementos HTML sobre los cuales se desea tener persistencia
- Gestionar los eventos (por ejemplo: OnClick) de los elementos HTML a nivel de JavaScript que hayan sido sobrecargados en el CGI

Para conseguir la persistencia de determinados elementos HTML y que por lo tanto alguna de sus propiedades sea visible desde el CGI se utiliza la cláusula de *Harbour* 'PERSISTENT' cuando se define la propiedad en su clase.

Con un ejemplo se ve más claramente:

```
CLASS ZEdit FROM WInput
  DATA cValue INIT "" PERSISTENT
ENDCLASS
```

Con la cláusula PERSISTENT a nivel de control HTML estamos indicando que queremos que la propiedad *'value'* de ese elemento HTML esté accesible en el CGI la próxima vez que se ejecute con el valor actualizado de la página web. La próxima vez que ejecute el CGI podrá comprobar como la propiedad *cValue* de ese elemento HTML ha recuperado de forma automática el valor que tenía en la página web.

Las clases de xaWeb que soportan los elementos HTML vienen prediseñadas con los valores PERSISTENT en los miembros de clase que en principio se estima es interesante tener persistencia. Si usted desea añadir persistencia en algún otro miembro deberá sobrecargar la clase Z indicando la cláusula PERSISTENT en los miembros adicionales que usted desee. Por ejemplo:

```
CLASS WEdit FROM WInput
  DATA lEnabled PERSISTENT
ENDCLASS
```

xaWeb incluye un mecanismo adicional para hacer persistente cualquier propiedad a **nivel de aplicación** y es utilizando la cláusula PERSISTENT a **nivel** de WDoc. Por ejemplo:

```
CLASS WDocMain FROM WDoc
  DATA cUserName INIT "" PERSISTENT
```

Internamente toda la sincronización de valores se realiza a través de cookies que se generan de forma automática. En el objeto '*Engine*' se crean varios hashes para este propósito, que son:

- *hState* que almacena todos los valores de elementos HTML tipo PERSISTENT.
- hEvent que almacena las propiedades del evento que se ha disparado desde la página web. Este *hash* que recibe el ID del control que ha disparado el evento, las coordenadas X e Y de pulsación, si las teclas Control y Alt estaban pulsadas al hacerse clic. Es posible que reciba información adicional referida al contexto donde se está usando.
- hCargo que almacena valores adicionales que pueden ser necesarios en algún tipo de evento. Por ejemplo, se utiliza en operaciones de edición de registros procedentes de una tabla HTML, en donde hCargo contiene los valores de las columnas de la tabla, que se pueden modificar y retornar para que sean actualizados en la tabla HTML.
- Y los ya comentados anteriormente de hParams y hPost que dan información sobre los parámetros pasados y los campos de un formulario en el caso de que se hubiese enviado.

El método ***Create*** que tienen todos los controles HTML de xaWeb es el responsable de asignar los valores PERSISTENT con la información del hash *hState* del objeto *Engine* y por lo tanto ha de ser ejecutado después de haber inicializado sus propiedades:

```
With Object WEdit():New( oSection )
  :cValue := "" // opcional
  . . .
  :Create()
End with
```

NOTA IMPORTANTE

Para que la persistencia de controles HTML funcione correctamente es necesario que los controles tengan asignada su propiedad cID y se ejecute su método Create().

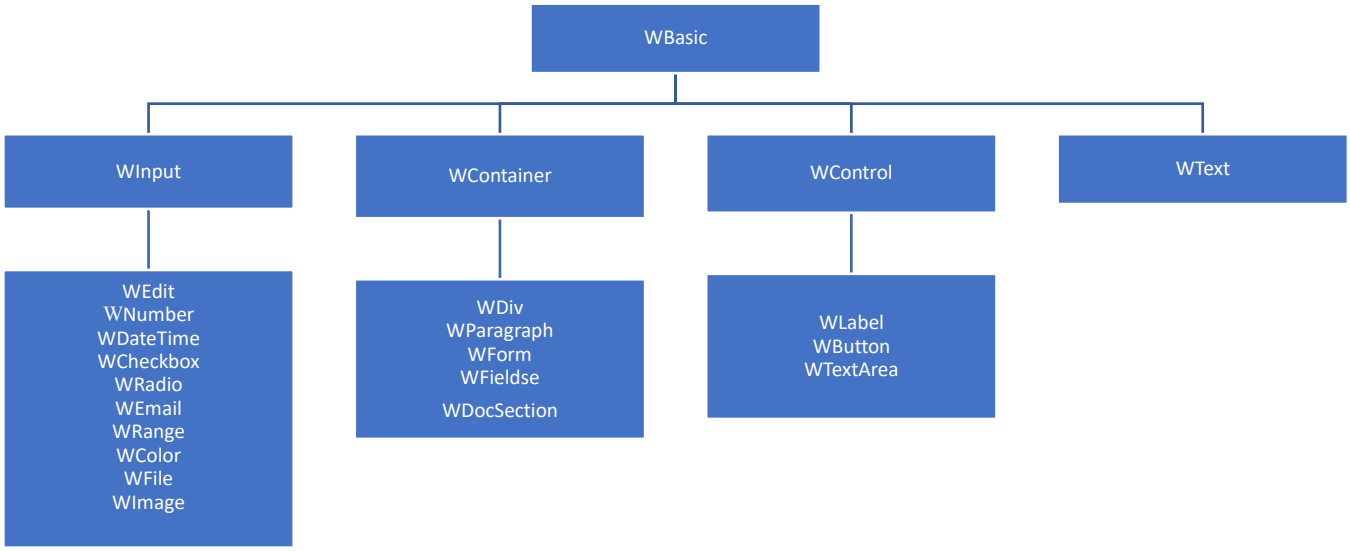
JERARQUÍA DE CLASES

Hasta ahora, estás son las clases que hemos comentado de xaWeb:

- Clase *WApplication* con su correspondiente variable pública *Application*.
- Clase *WEngine* con su correspondiente variable pública *Engine* que es la encargada de las comunicaciones con el servidor web.
- Clase *WRouter* que es la responsable de realizar el rutado inicial de la aplicación, estableciendo el tipo de operación a procesar, el objeto WDoc a instanciar y el método a ejecutar del mismo.
- Clase *WDoc* con su correspondiente variable pública *Document* que es la encargada de hacer todo el despliegue de HTML.

- Clase *WDocSection* que son objetos que representan las distintas secciones que puede almacenar un objeto *WDoc*.

En xaWeb cada elemento HTML es un objeto Harbour. Incluso para poner un simple 'Hola Mundo' es necesario crear un objeto especial de xaWeb y esto es debido a que el usuario NO DEBE enviar directamente contenido textual a STDOUT como por ejemplo hace PHP con el comando ECHO. El único responsable de realizar el despliegue del contenido HTML es el objeto *WDoc* que esté instanciado.



Esta es la jerarquía básica de clases de cualquier control HTML de xaWeb. Un control de edición como por ejemplo *WEdit* hereda de *WInput* y éste a su vez de *WBasic*. La clase *WContainer* engloba a todos los controles que pueden tener a su vez más controles HTML.

El objeto *WDoc* contiene una matriz de objetos *WDocSection* y cada objeto *WDocSection* contiene a su vez una matriz de objetos *aControls*, que si son del tipo *WContainer* pueden tener a su vez más controles. Cuando se solicita el despliegue HTML en WDoc a través de su método ***Render***, éste provoca el despliegue HTML en todas las secciones WDocSection, y cada sección hace a su vez lo mismo con sus matriz aControls, que pueden incluso tener más controles anidados.

Todo este proceso de despliegue se hace llamando a un método *RunHtml*() que tienen todos los controles y en todos los niveles de herencia. Por ejemplo: El método *RunHtml* de *WEdit* hará lo que tenga que hacer, luego llamará al método *RunHtml* de *WInput* y por último se llamará al método *RunHtml* de la clase *WBasic*.

Puede haber una relación **1-1** entre un elemento HTML y un control xaWeb. Por ejemplo, el elemento <h1> se corresponde con un único control *WText*() de xaWeb. Pero esto no es siempre así. Por ejemplo, el control *WTable* tiene múltiples controles hijos internos que son a su vez elementos HTML.

Esta relación se puede complicar aún un poco más cuando el control que representa el elemento HTML pasa a ser un control hijo de un control nuevo que se ha instanciado en su constructor. Con algunos ejemplos se ve más claro el concepto:

- Por defecto en HTML los elementos tipo <table> no tienen barras de desplazamiento vertical. Es decir, el elemento HTML ocupará verticalmente la suma de su cabecera, sus filas y su pie. ¿Cómo podemos conseguir que el control tenga una específica longitud vertical y nos muestre una barra de desplazamiento? Sencillo, incluimos el elemento <table> en un con dimensiones fijas o regladas vía CSS y que sea éste el que muestre la barra de desplazamiento. Por lo tanto, es necesario que cuando creemos un elemento <table>, se cree también un elemento que es el que realmente va a contener al elemento <table>. En el caso del control *WTable*, ese elemento es accesible a través de su propiedad *oContainer*.

- El control *WEdit* podría corresponderse con un único elemento HTML <input>, pero en xaWeb, tiene mucha más funcionalidad ya que incorpora el posible <label> y un mensaje de error cuando el valor introducido en el control no es válido. Cuando instanciamos un objeto WEdit realmente estamos instanciando un contenedor del tipo WDiv accesible como oContainer, que a su vez contiene un objeto WLabel accesible como oLabel para la descripción del campo, un objeto WSpan accesible como oError para mostrar el posible error y finalmente un objeto WInput tipo texto que es al que realmente se referencia en el retorno del constructor.

- Esta complejidad de múltiples controles instanciados de forma automática se complica aún más cuando se utilizan frameworks del tipo Materialize.

Esta creación de controles de forma automática puede resultar un problema, sobre todo, cuando se pretende iterar por todos los controles de un objeto *WDoc*; para evitar este problema todos los controles tienen dos propiedades que indican su ancestro heredado que son: **oParent** y **oOwner**.

La propiedad *oParent* establece la dependencia padre-hijo, de la misma forma que el HTML. Es decir, cuando creamos el objeto *WTable* heredando de un *WDiv*: El oParent de *WTable* sería *WTable:oContainer* y el parent de *WTable:oContainer* sería el *WDiv*. Como puede observar hemos introducido un objeto *oContainer* en medio de los dos. Para **iterar** sobre todos los controles, incluso los creados de forma automática y de la misma que el documento HTML, utilizaremos la matriz *aControls* que posee *WDoc* y todos los controles que heredan de *WContainer*.

Por el contrario, la propiedad *oOwner* indica cual es el control que le ha creado, bien por código escrito por el programador o de forma automática por xaWeb. Para **iterar** sobre este tipo de controles utilizaremos la matriz *aComponents* que posee *WDoc* y todos los controles que heredan de *WContainer*.

PREPROCESADO (PREVIO AL DESPLIGUE)

Como ya hemos explicado con anterioridad todo el despliegue del código HTML se produce cuando se ejecuta de forma automática WDoc:*Render* y éste ejecuta el método *RunHtml* en todos sus controles heredados, empezando por todas las secciones y éstas a su vez lo ejecutan en todos sus controles tipo elemento HTML, que a su vez pueden tener más controles.

Antes de producirse está ejecución en cascada del método *RunHtml*, se hace lo mismo con un método de nombre **PreProcess**. En consecuencia, antes de ejecutarse el método *RunHtml*, todos los controles procesan su método *PreProcess*, que existe en todos los niveles de herencia en la jerarquía de clases. Cada control ejecuta su método PreProcess, y cuando ha terminado llama a Super:PreProcess para seguir con su procesamiento en cascada. La clase *WBasic* que es la última en la jerarquía, y únicamente ejecuta el evento **OnPreProcess** que permite que el usuario añada todo el código que estime necesario.

Este método *PreProcess* es muy importante. En él, la mayoría de los controles realizan importantes modificaciones en el despliegue del control (y controles dependientes). Incluso pueden crear nuevos controles que también se desplegarán. Estos controles que se crean en este método están en ámbito en el objeto únicamente en el evento *OnPreProcess*.

TIPOS DE OPERACIONES DEL CGI

El CGI se comporta de manera distinta dependiendo de cómo haya sido ejecutado:

- Sin ningún tipo de parámetro: En este caso, el CGI generará la página Web del documento HTML que tenga definido por defecto.
- A través de parámetros que se pasan por comando tipo GET. Los comandos tipo GET que se envían en peticiones HTTP, son aquellos que se incluyen como texto adicional en la propia URL. Por ejemplo:

  http://localhost/test.cgi?**action**=mydoc-mymethod

  El primer parámetro pasado por comando GET establece el tipo de operación. En este caso sería "action" y su valor "mydoc-mymethod". Este ejemplo, pediría al CGI que instanciase el documento "mydoc" y ejecutase su método "mymethod". Como se trata de una operación tipo "action", sabemos que se ha producido dentro de un evento de nuestra página web y por ello recibiremos como parámetro en el método un *hash* con información sobre el evento: Quién lo disparo, coordenadas del ratón, estado de pulsación del teclado y algún dato adicional interesante. Dicho parámetro es realmente la propiedad *Engine:hEvent*.

  xaWeb recibe actualmente tres tipos de operaciones diferentes a través de comandos GET, que son:

1. Operaciones tipo “**Action**”, que son las que se producen por eventos en la página web, como la pulsación de un botón.

2. Operaciones tipo “**Form**”, que son las que se producen al realizar el ‘**submit’** de un formulario. En este caso el parámetro que se recibe es un *hash* con toda la información introducida en el formulario.

3. Operaciones tipo “**Service**”, que se producen al solicitar un servicio Web, bien de nuestra aplicación o de terceros.

4. Operaciones tipo “**Custom**”, que se producen al utilizar cualquier otro tipo de operación distinta de las tres anteriores. Deberá utilizar el hash *Engine:hParams* para saber exactamente la información que se ha pasado vía GET.

Todas las posibles operaciones son explicadas en profundidad en futuros capítulos.

PASO DE PARÁMETROS DESDE CÓDIGO

Ya hemos visto los tipos de operación que actualmente soportan los CGI de xaWeb, ahora vamos a ver cómo podemos establecer esos parámetros desde nuestro propio código. Los tipos de operación que el programador puede establecer directamente son ‘action’ y ‘service’, ya que el tipo ‘form’ se genera cuando se pulsa un botón ‘submit’ en un formulario. La forma manual de establecer un parámetro sería, por ejemplo:

```
oLink:hRef := “https://www.ejemplo.com/mi.cgi?action=mydoc-mymethod”
```

Que generaría el HTML para que, al pulsar el enlace, se cargará el CGI con los parámetros que hemos indicado. xaWeb posee dos métodos para establecer los parámetros de forma sencilla, que son:

- oWDoc:Action(<cMethod>, [<cDoc>])
- oWDoc:Service(<cMethod>, [<cDoc>])

```
oLink:cHRef := ::Action( “miMetodo” )
```

Aunque pudiera parecer que estos dos métodos retornan una cadena, realmente retornan objetos del tipo **WTask**, que ofrecen más funcionalidad que una simple cadena tipo URL, como por ejemplo, permite añadir parámetros adicionales en la URL con el método *AddParam(cName, xValue).*

La propiedad *WLink:chRef* es un método del tipo **SETGET** que siempre retorna una cadena, pero puede recibir una cadena o un objeto. El objeto **WTask** es almacenado en la propiedad *WLink:oHRef*.

Recuerde que en el caso de eventos sólo necesita indicar el nombre del método:

```
oBtn:OnClick := “miMetodo”
```

JAVASCRIPT

En xaWeb existe una clase exprofeso para la gestión de scripts de nombre **WScript**(), que se puede instanciar bien indicando una URL o el código directamente. Por defecto el script se ubicará en el '*footer*' de la página, pero con la propiedad ***lFooter*** a falso puede forzar a que el script vaya en el 'header'. Otra propiedad importante es ***lDefer*** que permite indicar que el script se debe de cargar una vez que la página haya sido completamente cargada. Esta propiedad realmente sólo la contempla la etiqueta '**script'** en el *header*, sin embargo, en xaWeb también tiene sentido usarla en scripts que vayan en el *footer* y esto es debido a la necesidad de poder controlar los scripts que han de ir antes o después de la *mochila* de código Javascript que añade xaWeb en todas las páginas. Ejemplos:

```
Document:AddScript( "http ….", cName ) → Objeto WScript()
```

Los objetos WScript tienen una propiedad de nombre '**cName**' que permite identificar cada uno de los módulos. Esta propiedad permite, si el programador lo desea, que todo el código del 'script' se recupere de un fichero externo en vez de que vaya incorporado dentro del CGI. Cuando un script tiene asignada esta propiedad *cName*, xaWeb buscará en el servidor la existencia de un script con ese nombre en la carpeta '**js**'. Si lo encuentra, incluirá una referencia a ese archivo, en vez de incluir todo el código del script.

También es posible crear código JavaScript directamente desde cualquier módulo de su aplicación y desplegarlo en el HTML resultante. Por ejemplo:

```
TEXT INTO cJs
  function myfunction(e) {
    alert('Click triggered from a user function');
  }
ENDTEXT

Document:AddScript( cJs )
```

CSS

Con la misma funcionalidad que se ha indicado para Javascript, xaWeb incorpora una clase exprofeso para la gestión de los CSS de nombre WCss que permite incluir referencias a ficheros CSS externos o introducir directamente código CSS.

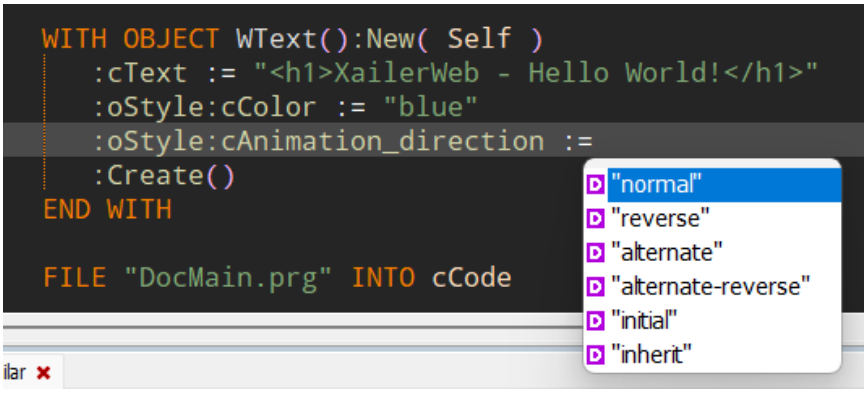
```
Document:AddCSS( cText, cName, lUri ) → Objeto WCss()
```

Los objetos WCss tienen una propiedad de nombre '**cName**' que permite identificar cada uno de los módulos. Esta propiedad permite, si el programador lo desea, que todo el código 'CSS' se recupere de un fichero externo en vez de que vaya incorporado dentro del CGI. Cuando un objeto *WCss* tiene asignada esta propiedad *cName*, xaWeb buscará en el servidor la existencia de un archivo CSS con ese nombre en la carpeta '**css**'. Si lo encuentra, incluirá una referencia a ese archivo, en vez de incluir todo CSS. Es posible añadir más código CSS con el método oCss:AddCode(cText).

xaWeb pretende seguir las mejores prácticas de programación y ello incluye el uso extensivo de CSS y anima a su utilización. No obstante, siempre es posible establecer un estilo directamente sobre el control con el método AddStyle(cText) que tienen todos los controles.

```
With Object WText():New( Document )
  :AddStyle( “color:red” )
  :cText := “Hello”
End with
```

Los estilos en línea también pueden ser introducidos a través de la propiedad oStyle que tienen todos los controles. En cuanto el usuario hace referencia a esa propiedad, de forma automática se instancia un objeto de la clase **WStyle**, que incorpora casi la totalidad de los estilos disponibles, y en los casos que sean enumerados, incluso le muestra una lista de los valores posibles:



El uso de estilos en línea con oStyle es totalmente compatible con el método AddStyle() y se pueden utilizar de forma conjunta, sin problema.

Además de esta propiedad oStyle, existe otra propiedad accesible en todos los controles de nombre oContext, que es una instancia de lo que se denomina un ‘*Context helper*’, es decir, una ayuda al paquete de contexto que esté utilizando, que se analizará en un capítulo posterior. Actualmente, el único ‘*Context package*’ que incluye un ‘*Context helper*’ es **Materialize.** Su uso es muy parecido a oStyle y al igual que él es soportado por el Intellisense de Xailer sin problema. El objeto oContext, es un simple asistente para que no tenga que recordar los nombres de clases y estilos que impone cada paquete de contexto. A modo de ejemplo (Materialize):

```
oDiv:oContext:Col( 12, 4, 2 )
```

Sería equivalente a establecer la clase “col s12 m4 l2”

En un capítulo posterior se explica con más detenimiento el sistema que utiliza xaWeb para integrar ‘*frameworks*’ como Materialize.

SERVICIOS WEB

Los servicios web permiten establecer comunicaciones con otras aplicaciones y recuperar cualquier tipo de dato. La solicitud del servicio se produce de forma asíncrona y sin abandonar la página web. Por lo tanto, es la página web la que debe de procesar la petición cuando ésta haya concluido. Para más información consulte el siguiente enlace.

xaWeb permite **consumir** cualquier servicio web de terceros e incluso comportarse él mismo como un servicio web. Observe que la opción de consumir se refiere al consumo en la propia página web y no al consumo desde el interior del CGI. Además, debemos de distinguir entre **consumo de servicios web de terceros** y **consumo de servicios propios de nuestro CGI** u otros CGI creados con xaWeb, ya que este último tipo ofrece mayor funcionalidad.

Para consumir cualquier servicio web de terceros tan sólo hay crear un objeto **WFetch** indicando la URL de la dirección web a la cual quiere acceder y asignarlo a un evento de un control. Cuando desde su página web se solicita un servicio web externo **de terceros**, lo más normal es recibir un objeto JSON, que puede procesar. Ha de tener en cuenta que el procesamiento se hace necesariamente en JavaScript y por lo tanto es necesario unos mínimos conocimientos de JavaScript para ejecutar servicios web de terceros.

Este pequeño ejemplo ejecuta un *Web service* de la web *ip-api.com* que nos permite saber la localización de nuestra IP pública:

```
oFetch := WFetch():New( "http://ip-api.com/json/example.com" )

WITH OBJECT oFetch
   :cTargetId := "mycity"
   :cSourceId := "button3"
   :cCallBack := "citySolver"
END WITH

WITH OBJECT WButton():New( Self )
   :cText := "This button calls an asyncronous API"
   :OnClick := oFetch
   :cId := "button3"
   :Create()
END WITH
```

El parámetro del constructor se asigna automáticamente al miembro *cUrl* del objeto. Los miembros que hay que asignar al objeto *WFetch* para este tipo de operaciones son:

- *cTargetID*: Con el ID del control que va a recibir la información
- *cSourceID*: Con el ID del control que dispara el evento. Esto se utiliza para dejar el control deshabilitado mientras se produce la ejecución del servicio web que es asíncrono. Opcional.
- *cCallBack*: Función JavaScript que recibirá el objeto JSON devuelto por el servicio web.

```
TEXT INTO cJs
   function citySolver(element, data) {
      if (data.city) {
          element.innerHTML = ‘Example server is at ' + data.city;
      } else {
        element.innerHTML = 'City could not be found';
      }
   }
ENDTEXT

Document:AddScript( cJs )
```

Es muy probable que el servicio web exija algún parámetro adicional vía **GET**, que consiste en incluir detrás de la URL, el carácter **‘?’** seguido de los parámetros en la forma: ‘key=value’ y separados cada uno de ellos por el carácter ‘**&**’. La forma más fácil de incluir dichos parámetros en la URL es utilizar un objeto **WTask** en vez de indicar la URL directamente, que se explica en un apartado posterior.

En el anterior párrafo hemos explicado como incluir parámetros dentro de la llamada al servicio web; pero si lo analiza detenidamente, se dará cuenta de que, en muchos casos es inútil, ya que los parámetros del servicio web dependerán de valores que el usuario introduzca en la página web, y, por lo tanto, valores que el CGI desconoce. No obstante, xaWeb ofrece una solución a este problema: Con xaWeb es posible el paso de parámetros de valores existentes en la página web. Deberá utilizar el método **WFetch**:AddJsParam(cId, cAtributo). ‘cId’ y ‘cAtributo’ se corresponden con el **ID** y el **atributo** del control HTML que desea pasar, que normalmente será ‘**value’**. Por ejemplo: **WFech**:AddJsParam(“idEdit”, “value”). Si se indica un atributo no estándar, xaWeb intentará encontrar dicho atributo como ‘**dataset’** del propio control. Para más información consulte este enlace.

El uso del control *WFetch* que hemos visto hasta ahora se caracteriza principalmente porque devuelve un *stream* de datos, que normalmente es del tipo JSON, aunque también podría ser XML o directamente un *stream* binario. Este tipo de operaciones FETCH son del tipo ***cContentType*** = *application/json;charset=utf-8*.

Si queremos que nuestro CGI se comporte como un servicio web o proveedor de servicios web tendremos que pasar como primer parámetro GET el comando “**service**” indicando el nombre de clase del documento **WDoc** y el método que desea ejecutar separados por un guion. A continuación de ese primer parámetro GET podrá incluir todos los que desee e incluso utilizar el método POST de forma adicional si así lo desea. Por ejemplo:

http://midominio.com/miapp.cgi?service=mydoc-mymethod

NOTA IMPORTANTE

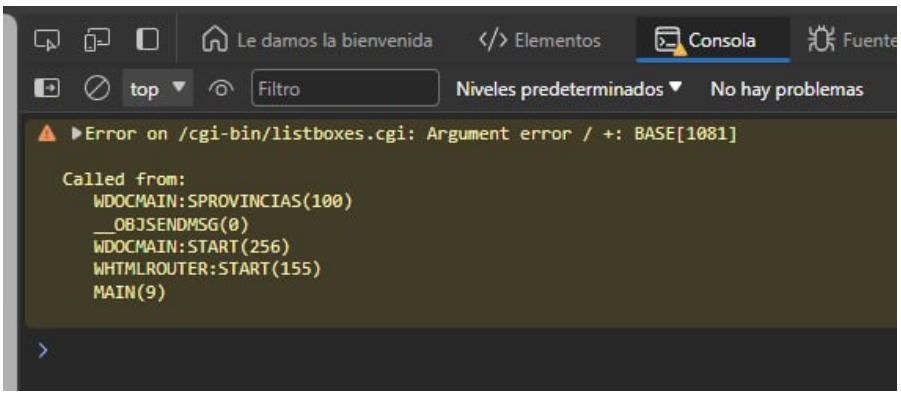
xaWeb puede funcionar como un servicio web, pero hay que tener en cuenta que cada vez que se ejecuta un servicio xaWeb, se ejecuta nuestro CGI, se ofrece el servicio y automáticamente se **sale** de la aplicación. Cualquier modificación que haya hecho usted en su programa, se pierde. Ha de pensar que realmente no se está ejecutando su programa, si no más bien, un servicio aparte que ofrece su programa, pero que nada o casi nada tiene que ver con él.

El único parámetro que recibirá el método es un *hash* con todos los parámetros enviados. El método debe de retornar un *'stream'*. Normalmente se tratará de una cadena tipo JSON (cContentType = application/json), pero también puede ser directamente código JavaScript que será interpretado por nuestra página web como veremos más adelante.

xaWeb incorpora un potente mecanismo para evitar que haya que procesar desde la página web el valor retornado por el servicio web, pero este mecanismo sólo funciona con servicios web ofrecidos por CGIs realizados con xaWeb y consiste en retornar directamente código JavaScript que será procesado de forma directa por la página web cuando la petición se reciba. Sólo hay que establecer la propiedad cContentType del objeto *WFetch* a "***application/javascript***" y lógicamente ya no es necesario indicar la propiedad *cCallBack*.

Las modificaciones que realice en cualquier propiedad de los controles HTML dentro de su CGI, de forma interna se convertirán en instrucciones JavaScript que se ejecutarán en la página web.

Cuando el error se produce dentro de un 'servicio' (tipo "service") no existe una página web en la cual se pueda mostrar la información de error. En dicho casos, los errores se mostrarán únicamente en la consola del navegador con mensajes del tipo 'console.warn(…)'.



LA CLASE WTASK

Esta clase permite manejar de forma ágil y sencilla las URL que va a exigir nuestra aplicación. Dependiendo de en que contexto se use se debe utilizar un constructor u otro:

- Para indicar una operación tipo "*action*" a un método de cualquier documento, utilizaríamos el constructor WTask:Action(<cMethod>, [<cDocument>]), donde cMethod es el nombre del método a ejecutar y cDocument es el documento a cargar, que por defecto será el que se establezca por defecto en la aplicación.
- Para indicar una operación tipo "*service*": WTask:Service(<cMethod>, [<cDocument>])
- Para indicar una operación tipo "*Custom*": WTask:Custom(cOperation, cDocument)
- Para indicar directamente una URL utilizaremos: WTask:Url(<cUrl>)

El objeto **WTask** recibido se debe asignar directamente, de la misma forma que asignaría la propiedad de la URL. Por ejemplo `oLink:cHref := oTask, oButton:OnClick := WTask:Action("miMetodo")`

La clase **WTask** incluye el método AddParam(<ckey>, <cValue>), que permite añadir fácilmente parámetros del tipo GET a la URL y el método SetParam(<nPos>, <cValue>) para cambiar el valor de un parámetro cualquiera.

Es posible asignar directamente a un control tipo **WLink**, por ejemplo, el texto resultante de la URL que contiene el objeto **WTask** con el método WTask:Html().

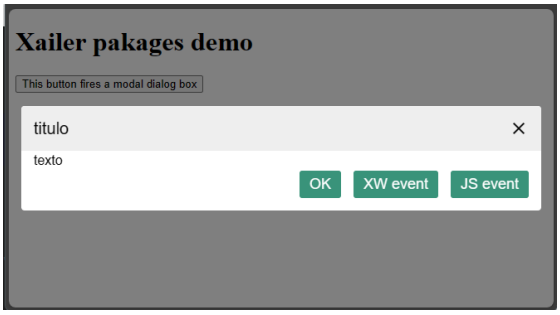
PAQUETES

La gestión de paquetes de xaWeb es una herramienta super potente que permite ampliar de forma muy sencilla la funcionalidad de las páginas web. Los paquetes no son más que un envoltorio de código CSS y JavaScript que tiene la funcionalidad extra de crear objetos ad-hoc para dicho paquete que son capaces de relacionarse con xaWeb. Se ve más claramente con un ejemplo:

```
oPackage := WModal():New( Self )
oModal := oPackage:ShowModal( "Artículos", "Confirme borrado", { "Si", "No" } )
oModal:OnClick( 1, "BorrarArticulo" )

WITH OBJECT WButton():New( Self )
  :cText := "Borrar artículo"
  :OnClick := oModal
  :cId := "button1"
  :Create()
END WITH
```

Este ejemplo muestra un diálogo modal que si se pulsa el botón 'Si' ejecuta el método 'BorrarArticulo' del documento. Para más información ver ejemplo 10-Packages.



Todos los paquetes creados para xaWeb deben de heredar de la clase ***WPackage*** que incluye la funcionalidad mínima que ha de tener un paquete, que básicamente consiste en añadir CSS y JavaScript, bien a través de URL o directamente por código.

Las clases que heredan de ***WPackage*** han de permitir instanciar objetos que dependan de él mismo y han de incorporar un método **Html**. Eso es todo. Añadir cualquier componente web gratuito o comercial a través de paquetes es una tarea muy sencilla.

CONTEXT PACKAGES (PAQUETES DE CONTEXTO)

Los paquetes tipo '**Context'** son paquetes especialmente diseñados cuando se decide utilizar xaWeb con un '***framework'*** adicional. Estos paquetes son los encargados de configurar todos los colores y CSS general que utilizará la aplicación. Los paquetes se han diseñado para que sean fácilmente sobrecargados por el programador y pueda adaptarlos a su propio gusto. Simplemente incorporando el paquete en nuestro documento, se modificará completamente su aspecto.

Actualmente xaWeb incluye cuatro paquetes:

1. **WaterContext**: Basado en Water.CSS que está pensado para hacer páginas web sencillas, más bonitas, responsivas y con control de temas. Recomendamos su uso cuando la vistosidad premium no sea importante.

```
WITH OBJECT WWaterContext():New( Self )
   :cTheme := "auto"
END WITH
```

2. **SimpleContext**: Basado en Simple.CSS que tiene el mismo propósito que el anterior, pero con otro estilo. Este estilo tiene soporte de temas de forma automática, pero no permite cambiarlo fácilmente por código.

3. **Basic**Context: Este paquete está pensado para que sea el propio usuario el que diseñe las características principales de los colores que utilizará su aplicación, tanto en modo claro, como oscuro y establezca los valores CSS por defecto de cada uno de los Tags que utilizará su aplicación. Es completamente personalizable.

4. **MaterializeContext**: Este paquete incluye el framework completo de Materialize, que no sólo incluye gestión de temas y responsividad, sino que incluye gran cantidad de componentes y controles propios que facilitan mucho la creación de páginas web.

Todos los paquetes de contexto que incluye xaWeb son configurables por el programador a través de variables CSS y su propiedad *WContext:cCssMods* que permite sobrecargar los valores iniciales CSS del paquete. Cada paquete de contexto utiliza sus propias variables que inicializan a los valores que ellos estiman oportunos, pero que se pueden cambiar sin problemas.

Desgraciadamente, cada paquete utiliza sus propias variables y el desglose de los colores disponibles varía mucho entre paquetes. En xaWeb hemos querido unificar los colores que puede utilizar una aplicación web. Independientemente de que cada paquete utilice sus propias variables, con más o menos profundidad en el desarrollo de estas, hemos establecido los colores básicos que debe de tener una aplicación web de gestión empresarial, que serían los siguientes:

1. Color de fondo, fondo alternativo, texto y texto rebajado (mutted) de la página (body)
2. Color de fondo, texto y 'hover' primario
3. Color de fondo, texto y 'hover' secundario
4. Color de fondo, texto y 'hover' destacado (accent)
5. Color de borde
6. Color de gradiente

El color de fondo alternativo de página se utilizaría para paneles, tarjetas y efectos de deshabilitado; el color primario para enlaces, botones, efecto de foco, y estados activos de formulario; el color secundario para cabeceras (headers) y pies de página (footers), el color destacado para controles y áreas que se deseen destacar, el color de borde se usa para líneas y bordes y por último el color de gradiente se utiliza para establecer el efecto de opacidad/transparencia.

Independientemente del paquete de contexto utilizado, todos estos colores son accesibles por parte del programador a través de métodos de la clase WContext:

```
METHOD BodyColor()            INLINE "var(--body-color)"
METHOD BodyAltColor()         INLINE "var(--body-alt-color)"
METHOD BodyTextColor()        INLINE "var(--body-text-color)"
METHOD BodyTextMutedColor()   INLINE "var(--body-text-mutted-color)"
METHOD PrimaryColor()         INLINE "var(--primary-color)"
METHOD PrimaryHoverColor()    INLINE "var(--primary-hover-color)"
METHOD PrimaryTextColor()     INLINE "var(--primary-text-color)"
METHOD SecondaryColor()       INLINE "var(--secondary-color)"
METHOD SecondaryHoverColor()  INLINE "var(--secondary-hover-color)"
METHOD SecondaryTextColor()   INLINE "var(--secondary-text-color)"
METHOD AccentColor()          INLINE "var(--accent-color)"
METHOD AccentHoverColor()     INLINE "var(--accent-hover-color)"
METHOD AccentTextColor()      INLINE "var(--accent-text-color)"
METHOD BorderColor()          INLINE "var(--border-color)"
METHOD GrColor()              INLINE "var(--gr-color)"
```

Para cambiar cualquier color en **WBasicContext**(), tiene cuatro opciones:

1. Modificar manualmente el archivo CSS: *xw_BasicContext.css*
2. Establecer los nuevos colores en las matrices **WContext**:aDarkColors y **WContext**:aLightColors

```
DATA aDarkColors  INIT {"#00001a","DarkSlateBlue","white","LightGray",;
                        "Navy","cornflowerblue","white",;
                        "MediumVioletRed","HotPink","DimGray",;
                        "Gold","PaleGoldenrod","Black",;
                        "LightSteelBlue", "64"}

DATA aLightColors INIT {"white","whitesmoke","black","gray",;
                        "Tan","Wheat","black",;
                        "Teal","LightSeaGreen","DimGray",;
                        "SkyBlue","LightBlue","Black",;
                        "Bisque", "255"}
```

3. Utilizar la propiedad cCssMods para sobrecargar los valores que desee. Por ejemplo:
   `--body-color: white;`
4. Cambiar el color utilizando el método **SetColor**(cName, cValue, lDark)

Todos estos cambios son a nivel de **contexto** de toda la aplicación. Si requiere cambiar el color de un simple control recuerde que sólo debe hacer: `-oButton:AddStyle( “color: White;” )`

En su código CSS utilizará por ejemplo: var(--body-color), mientras que a nivel de CGI puede utilizar oContext:BodyColor(). Estos colores sólo existen en su totalidad para el paquete de contexto W**Basic**Context(). Es muy probable que otros paquetes de contexto no soporten algunos de estos colores y su uso sea inútil. Recomendamos comprobar en el código de cada paquete de contexto, los colores que son soportados.

El paquete básico de contexto (WContext), del cual heredan el resto de los paquetes de contexto, es el responsable de establecer el tema de visualización: *‘none’, ‘light’, ‘dark’* o *‘auto’* con su propiedad cTheme. En cuanto un ‘*tema*’ ha sido establecido, el paquete guarda esa información con el nombre ‘theme’ en ‘Session storage’, en una ‘Cookie’ y en un atributo global del documento (document.documentElement). Dicha información es accesible desde cualquier paquete de contexto.

También es posible crear innumerables colores de usuario desde cualquier paquete de contexto con el método WContenst:AddCustomColor(cName, xLightColor, xDarkColor), que luego se pueden asignar a cualquier control:

```
oControl:oStyle:Color := ::oContext:CustomColor( “my-color” )
```

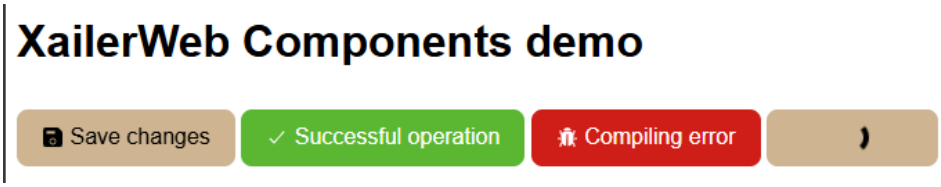
PDF PACKAGE

xaWeb incorpora un paquete específico para la impresión en formato PDF de cualquier tipo de documento que es realmente fácil de utilizar. Este paquete utiliza un sistema de plantillas para crear fácilmente cualquier tipo de informe. Actualmente (Oct 2024) se soporta la impresión automática de cualquier tabla HTML. En breve se incorporará la plantilla de 'facturas' que ofrece también dicha librería.

COMPONENTES WEB

Los componentes web son bloques de código que encapsulan la estructura interna de elementos HTML, incluyendo CSS y JavaScript, permitiendo así que el código se pueda volver a usar como se quiera en otras webs y aplicaciones. xaWeb permite la utilización de cualquier componente web de terceros, facilita la creación de componentes web y simplifica al máximo su uso. Puede ver un ejemplo en "samples\Comp-buttons". En él podrá observar los controles **WCmpButtonIcon, WCmpButtonSpinner Y WCmpNumericKeypad**.

Los componentes Web tienen la gran ventaja de que encapsulan todo el código necesario (incluido JavaScript y CSS), **por el contrario**, complica el acceso a los posibles elementos internos del componente. Por lo tanto, su uso debe hacerse únicamente en casos muy claros que un componente es lo más deseado. El ejemplo que damos es un buen punto de partida para que la comunidad de xaWeb empiece a realizar sus propios componentes y los haga accesibles a la comunidad o los venda por Internet.



EL CONTROL TABLE

Hemos prestado especial atención al componente de tabla de HTML, incluyendo en él y de forma opcional todo lo que un programador xBase está acostumbrado a encontrar en este tipo de control, pero sin descuidar la posibilidad de establecer cualquier configuración especial que se necesite en las tablas. Recomendamos tener un mínimo de conocimientos sobre el control HTML Table antes de proseguir la lectura.

El sistema básico de HTML de ir definiendo todas las filas de la tabla es inusual en xBase y poco práctico. Es preferible cargar la información con el método LoadData() que incluye la clase ***WTable***. Este método recibe una matriz con todas las filas del control. Cada fila es una matriz de columnas. Este método se puede llamar múltiples veces. De hecho, lo normal es llamarlo una primera vez para

establecer la cabecera de las columnas (*header*), una segunda vez para establecer los datos (*body*) y una última vez para establecer los pies de columnas si existiesen (*footer*). Las propiedades *nHeader* y *nFooter* aceptan un valor numérico en el que se indica el número de filas de que consta cada una de esas secciones.

El control **WTable** utiliza objetos internos para guardar toda su información, para que sea posible establecer nuevos estilos o propiedades a cada una de las secciones que puede tener un HTML Table, como son:

- *aHeaders*: Matriz de objetos *sTableCol* que representan la cabecera de la tabla
- *aRows*: Matriz de objetos *sTableRow* que representan cada una de las filas de la tabla
- *aColGroup*: Matriz de objetos *sTableColGroup* necesarios para establecer las propiedades HTML ColGroup
- *oHeader*: Objeto sTableZone que define la zona de cabecera de la tabla
- *oFooter*: Objeto sTableZone que define la zona de pie de la tabla
- *oBody*: Objeto sTableZone que define la zona de datos de la tabla

Es posible acceder a cada celda de la tabla. Por ejemplo:

```
oTable:aRows[2]:aCols[1]:nRowSpan := 2
```

Los eventos OnStartRow(oSender, oRow, nType) y OnStartCol(oSender, oCol) le permiten acceder a cada una de las filas y celdas antes de que se genere su código HTML para poder establecer cualquier estilo o propiedad en cada una de ellas. Estos eventos sólo se disparan cuando alimenta la tabla desde el CGI con una operación tipo '**action'**. Las operaciones tipo '**Fetch'** que devuelven directamente un JSON a nuestra página web no tiene mucho sentido que se disparen dichos eventos ya que no es posible realizar ninguna operación en el DOM de la página.

Por defecto en HTML, no se establece un largo máximo en las tablas. Todas sus filas se muestran de forma consecutiva y es en la propia página Web donde se establecen las barras de desplazamiento y no dentro del 'browse'. Para conseguir que la longitud de la tabla tenga una altura máxima hay que introducirla dentro de un contenedor (div) que tenga limitado su altura. Existen otras formas de hacerlo, pero en mi opinión, esta es la más recomendable. Por dicho motivo, el elemento WTable siempre tiene un contenedor, que puede ser accedido con su propiedad **WTable**:oContainer. Por lo tanto, el oParent de un objeto **WTable** es siempre su objeto oContainer.

Para poder realizar las clásicas operaciones de altas, edición y bajas es necesario tener una referencia a cada una de las filas de **WTable**. xaWeb utiliza una técnica muy sencilla que consiste en utilizar la primera columna de la tabla como identificador de cada una de las filas. El método LoadData() recibe un segundo parámetro que permite indicar si la primera columna es un identificador de fila o no. La propiedad lShowID (por defecto a falso) le permite indicar si desea que esa columna sea o no visible.

Al objeto WTable se le han añadido una serie de propiedades para simplificar su uso, que son:

- *cHeaderBkColor*: Color de fondo de la cabecera

- *cHeaderColor*: Color del texto de la cabecera
- *cFooterBkColor*: Color de fondo del pie
- *cFooterColor*: Color del texto del pie
- *lResponsive*: Para que la tabla sea responsiva y se muestre como tarjetas en dispositivos móviles
- *lCanSort*: Para que la tabla permita ordenar por cualquier columna
- *lCanFilter*: Para que la tabla permita filtrar por cualquier columna
- *lShowSelected*: Para que se muestra la fila seleccionada

Dos propiedades que merecen mención aparte son *cBtnDelId* y *cBtnEditId*. Estas dos propiedades permiten indicar el ID de los posibles botones que pudieran existir para borrado y edición de registros. Los botones permanecerán deshabilitados hasta que haya seleccionado una fila de la tabla. La propiedad *lShowSelected* debe de tener un valor verdadero.

FORMULARIOS

xaWeb ha simplificado al máximo el uso de formularios. Sólo es necesario indicar el método a utilizar y establecer su propiedad **cName** que ha de coincidir con el nombre de un método de su objeto ***WDoc***. La propiedad 'cAction' no es necesaria establecerla, salvo que quiera llamar a otra URL que nada tenga que ver con xaWeb.

xaWeb ha optado por no tener un control específico INPUT type-button ya que es lo mismo que el elemento HTML <button>, pero con menor funcionalidad ya que no permite establecer etiquetas HTML en su texto. Por dicho motivo es importante que cuando incluya un botón en su formulario, éste tenga su propiedad cType al valor "submit".

Los botones en HTML tienen una propiedad de nombre 'type' que permite establecer el tipo del botón. Estos son:

- button
- submit
- reset

xaWeb incorpora un tipo más de nombre '***cancel***' utilizado para establecer de forma automática la acción a realizar cuando se pulsa. De esta forma el programador se desentiende de tener que asignar su evento *OnClick*. Finalmente, los botones tipo '*cancel*' se generan como de tipo '*button*'.

El evento que dispara la aceptación del formulario es el evento ***OnSubmit*** del propio formulario. Cuando un control tipo **WButton**, su tipo está indicado como '*submit*'; al ser pulsado, además de su propio '*OnClick*' se dispara el evento ***OnSubmit*** del formulario, si estuviese sobrecargado.

Por defecto, este tipo de formularios provocan una operación tipo 'FORM' lo que implica la llamada a una nueva URL desde el navegador y por lo tanto la pérdida de todo el contenido de la página actual. xaWeb incorpora también la opción de poder enviar formularios a través de procesos FETCH

y por lo tanto no se sale de la página actual. Este tipo de formulario es explicado en un capítulo posterior.

xaWeb tiene un control por cada tipo de elemento HTML **Input**. Por ejemplo, **WEdit** se corresponde con un input tipo 'text' (text, password, tel, url y search). Existe un control **WEmail** que se corresponde con un input tipo "email", y así con todos los controles tipo **input**. En programación HTML es necesario crear un control adicional tipo "label" para asignar a cada elemento input, y, de hecho, si no se crea este control, el navegador suele indicar un aviso por su ausencia. xaWeb crea automáticamente dichos controles tipo 'label' de forma automática. Ello no significa que no tenga acceso al mismo. De hecho, puede establecer cualquier estilo, clase o lo que desee a ese control **Wlabel**. Todos los controles tipo 'input' tiene la propiedad ***oLabel*** (con la clase 'xw-input__label'). De la misma forma que se contempla un 'label' automático, xaWeb también incluye un '**div'** para indicar un posible error de validación de nombre *oError* (con la clase 'xw-input__error') y que por supuesto también es accesible para modificar su estilo. Cuando se crea un input en xaWeb, realmente se crea el input y tres controles más, que son:

1. Un **Div** que engloba a todos los controles, incluido el input y cuyo padre es el oParent indicado en la creación del control input (clase 'xw-input')
2. El elemento 'label' cuyo **oParent** será el '**div'** creado en el punto 1
3. El elemento 'input' propiamente dicho, cuyo **oParent** será el '**div'** creado en el punto 1 (clase 'xw-input__input')
4. El elemento '**div'** para mostrar el posible error, cuyo **oParent** ser el '**div'** creado en el punto 1

Las clases establecidas en los controles le permiten establecer su aspecto fácilmente a través de CSS.

Un caso especial, y al igual que ocurre en Xailer, es el control **WRadMenu** que es un único control que engloba varios 'radio-buttons' que funcionan de forma conjunta. Con un simple control se crea toda la estructura indicada anteriormente por cada uno de los elementos tipo 'radio' del control. El control permite alineación vertical u horizontal y su uso es muy sencillo.

TIPOS DE OPERACIÓN FORM

Este tipo de operaciones se produce cuando se pulsa el botón '**Submit'** de un formulario. La propiedad '**cName'** del formulario indica el nombre del método que se ha de ejecutar en el CGI. Este tipo de operaciones provoca una recarga de la página web incluyendo la información del formulario:

```
http://example.com/test.cgi?form=wdocmain-myform
```

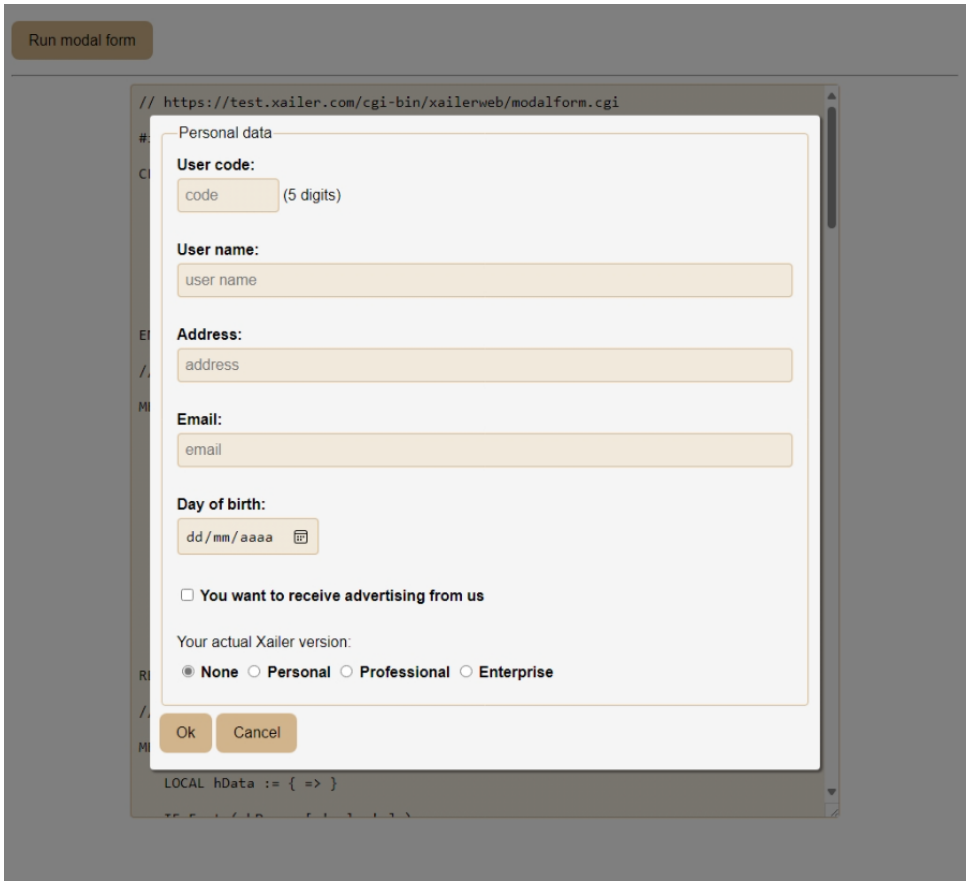
El método recibe como único parámetro un hash (hPost) con toda la información de los campos del formulario.

USO AVANZADO DE FORMULARIOS

Los formularios que hemos visto hasta tienen dos características de simplicidad:

- No se muestran por encima de la página web existente
- La invocación, aceptación (pulsar OK) o cancelación (pulsar Cancelar) del formulario exige una recarga de la página original indicando una determinada acción (action)

La clase **WForm** tiene una data de nombre ***lModal*** que le permite mostrar el formulario de forma flotante y centrado por encima de la página web.



Esta propiedad ofrece un gran aspecto visual al formulario, pero como éste va centrado en la pantalla (viewport) es probable que quede recortado en dispositivos móviles si el formulario es muy largo y entonces se muestren barras de desplazamiento vertical.

Para mostrar el formulario por encima de la página se puede realizar de dos formas:

1. Ejecutar un proceso '**action'** que cargue el formulario añadiendo una nueva **wDocSection** cuando se produzca un evento (ejemplo 13-ModalForm)
2. Cargar el formulario siempre, pero oculto, y sólo mostrarlo cuando sea necesario (ejemplo 14-ModalFormFetch). En este caso se carga el formulario junto con la sección principal en una nueva **WDocSection**, poniendo su propiedad lDeploy y lHide a verdadero. El botón que muestra el formulario deberá llamar en su evento OnClick a método **ShowSection**(cName) que tienen todos los controles pues está definido a nivel de la clase **WBasic**.

Con la primera opción se puede perder el estado de la actual página web ya que se realiza la carga de una nueva URL por parte del navegador. Sin embargo, con el segundo método no se pierde. La acción del evento Onclick del botón 'Cancel' que haya incorporado en el formulario con el tipo 'cancel' (ctype='cancel') se ajustará de forma automática dependiendo de la forma que haya optado para mostrar el formulario.

Todos los controles tipo '**input'** tienen un evento adicional de nombre '**OnValidate'** que permite validar el texto existente en cada uno de ellos. Este evento se ejecuta automáticamente cuando el control cambia su valor, pero también cuando se intenta hacer un 'submit' del mismo. El evento, al igual que en el resto de los casos de xaWeb se puede procesar a nivel de JavaScript o a nivel de CGI:

- Si se resuelve a nivel de JavaScript recibirá en la función que haya indicado dos parámetros: el **valor** del control y el **elemento** HTML. Deberá retornar un simple JSON con la propiedad 'pass' con verdadero o falso y otra propiedad 'error' (opcional) con la descripción del error producido.
- Si se resuelve a nivel del CGI (Harbour) mediante una operación tipo '**service**', recibirá en su único parámetro **hParam** los valores: **'Value'** e '**Id'**. Deberá retornar un simple JSON con la propiedad 'pass' con verdadero o falso y otra propiedad 'error' (opcional) con la descripción del error producido. La forma más sencilla de hacerlo es creando un Hash con las claves 'pass' y 'error' y luego retornar el JSON con la función **HB_JsonEncode(** hHash **)**. Es posible que reciba también otro valor en el hash de nombre "append_mode", que puede ser verdadero o falso. Esto sólo ocurre cuando se conecta una tabla con un formulario. Opción que se comenta más adelante.

En el ejemplo 14-ModalForm**Fetch**, además de utilizar el sistema de carga oculta del formulario, incluye una funcionalidad extra, que es la de realizar todo el proceso mediante una operación tipo 'service', lo que permite que nunca se abandone la página web realmente. Para conseguir esta funcionalidad tan sólo hay que sobrecargar el evento OnSubmit del formulario utilizando el método **SubmitToService(** cMethod **)** del propio formulario:

```
oForm:OnSubmit := oForm:SubmitToService( "MyFormData" )
```

El método **SubmitToService**(cMethod, lJson) indica el método de nuestro CGI que se debe de ejecutar cuando se haya aceptado el formulario y si éste debe de devolver o no, un objeto JSON para ser procesado por xaWeb después de la edición. Por defecto lJson es falso, que es el caso más habitual cuando **simplemente** se desea actualizar el contenido HTML de un elemento de la página web, como es el caso del ejemplo 14-ModalFormFetch. Cuando **no** retorna un objeto JSON, la operación Fetch se hace del tipo "application/javascript" y por lo tanto nuestro servicio se encargará de forma automática de generar el código JavaScript para que los elementos que se han modificado en nuestro CGI sean también modificados en la página web.

Al tratarse de un servicio, el método recibe un parámetro del tipo hParam. Es decir, un Hash con los parámetros enviados en la operación **Fetch**. Sin embargo, al tratarse de un formulario, lo normal es

que esa información se envíe a través de POST, por lo que el parámetro recibido es inútil. Esto no supone ningún problema, ya que toda la información POST puede conseguirse de la propiedad **Engine:hPost**.

La clase formulario tiene un evento adicional de nombre '**OnFormShow'**, que se usa básicamente para poder establecer valores iniciales en cada uno de los controles tipo Input que tenga el formulario. Este evento cuando es asignado con un literal coincidiendo con el nombre de un método del documento, dispara un proceso tipo 'Servicio' (operación Fetch) ejecutando dicho método. La propiedad Engine:hCargo contiene un objeto Hash con todos los ID de los controles tipo input y su actual valor. Sólo tendrá que modificar los que desee y retornar de nuevo el hash utilizando la función hb_JsonEncode(hHash). Puede ver un ejemplo de uso de este evento en Samples\16-Form-init.

USO DE MÁSCARAS EN INPUTS

Una importante funcionalidad que se echa de menos cuando abandonamos el lenguaje Harbour, es el uso de máscaras con la cláusula 'Picture' que tienen los controles estándar tipo GET o en el caso de Xailer, la propiedad cPicture que poseen muchos de los controles de edición.

En xaWeb se ha incorporado una librería para poder disponer de máscaras en los controles tipo <**input.text**> con una funcionalidad muy parecida a la que existe en Harbour. La librería seleccionada ha sido Maska, pero al igual que ocurre con las sesiones, se permite que el usuario seleccione cualquier otra librería para este propósito. El gestor interno de máscaras por defecto se recupera internamente llamando a la función **InputMask**() que retorna un "singleton" del gestor, que por defecto, es una instancia de la clase **WInputMask**.

Los controles tipo WEdit incorporan una propiedad de nombre '**cPicture'** que permite establecer la máscara del control. Estas son los distintos elementos de plantilla que acepta la propiedad:

1. Elementos propios de *Maska*:
    a. '#': Dígitos del 0 al 9
    b. '@': Letras sin incluir los caracteres internacionales
    c. '*': Dígitos y letras sin incluir los caracteres internacionales
2. Elementos de xaWeb al estilo de CA-Clipper (un elemento se corresponde con un carácter introducido):
    a. 'A': Letras incluyendo los caracteres internacionales
    b. 'N': Dígitos y letras incluyendo los caracteres internacionales
    c. 'D': Dígitos
    d. 'U': Letras incluyendo los caracteres internacionales en mayúsculas
3. Elementos de xaWeb distintos de CA-Clipper (un elemento se corresponde con uno o más caracteres introducidos)
    a. '0': Dígitos
    b. '9': Dígitos opcionales

c. ‘B’: Dígitos y letras incluyendo caracteres internacionales
d. ‘V’: Dígitos y letras incluyendo caracteres internacionales en mayúsculas

Ejemplos:

- "0.99": Numérico de cualquier longitud y dos decimales
- "#99.#99.#99.#99": Dirección IP
- "B B": Dos palabras
- "V V V": Tres palabras en mayúsculas

4. Elementos de xaWeb tipo función, únicamente para valores numéricos:

   a. “!#ll:dd:p” Donde:
      - 'll' son el identificador local del formato. Por defecto el indicado en InputMask():cLocale
      - ‘dd’ es un valor numérico con el número de decimales a utilizar
      - ‘p’ es un valor numérico que si distinto de cero, sólo admitirá valores sin signo

TABLAS Y FORMULARIOS: GESTIÓN COMPLETA

Se ha hecho un importante esfuerzo para que las operaciones tipo CRUD que se apoyen en tablas HTML sean fáciles de usar. La complejidad que supone la comunicación entre CGI y página Web, sin perder la información de la página web, obliga a realizar todas las operaciones sin recargar la página y haciendo que todos los procesos de altas, modificaciones y supresiones se realicen a través de *Web services* que proporciona el propio CGI.

La gestión conjunta de tablas y formularios para hacer un completo CRUD conlleva cierta complejidad cuando se pretende hacer de forma completa con servicios **Fetch**. Es decir, sin tener que recargar la página web. Estos serían los pasos necesarios:

1. Crear el formulario como se indicó en el ejemplo 14-ModalFormFetch. Es decir, cargando la sección del formulario de modo oculto y utilizando la propiedad WForm:lModal a verdadero.
2. Mostrar el formulario asignado al evento OnClick del botón que lo dispare el método WButton:ShowSection(cSection).
3. Indicar en el evento OnSubmtit de **WForm** el método de nuestro **WDoc** que queremos que se disparé tanto para altas como para edición: **WForm:SubmitToService**("Srv_Method", .T.) pasando como parámetro adicional un valor lógico **.T.** que indica que vamos a **retornar** un objeto JSON en el método.
4. Establecer la propiedad cTableId de **WForm** al identificador de la tabla que se desea procesar
5. Establezca la propiedad lShowSelected de **WTable** a verdadero para que sea posible seleccionar una fila

6. **Opcional**: Si tiene creados botones para 'Editar' y 'Suprimir' puede establecer las propiedades cBtnEditid y cBtnDelId con los identificadores de ambos botones. De esta forma se deshabilitarán automáticamente cuando no haya una fila de la tabla seleccionada.
7. Establezca la propiedad cDataField de todos los controles tipo **WInput**. El valor ha de coincidir con el valor que dio a los campos en la tabla.
8. Si existe algún campo que no desee que se pueda editar en modo edición, establezca la propiedad lDisabledOnEdit de dicho campo a falso.
9. Establezca en el evento OnClick del botón que realiza el alta el método **Append**(cFormSection) de **WTable**. Recibe como único parámetro el nombre de la sección donde se encuentra el formulario.
10. Establezca en el evento OnClick del botón que realiza la edición el método **Edit**(cFormSection) de **WTable**. Recibe como único parámetro el nombre de la sección donde se encuentra el formulario
11. Para la supresión de un registro no necesita mostrar ningún formulario, tan sólo es necesario llamar al servicio que lo elimina. En el ejemplo 15-Tables CRUD hemos optado por mostrar un mensaje de confirmación apoyándonos en el paquete **WModalMsgBtn**.

El método asignado en el evento OnSubmit de **WForm** será el responsable de validar la operación de **alta** y **edición**, y le permitirá hacer todas las operaciones internas en su base de datos. En este método recibe toda la información en **Engine**:hPost. En concreto:

- Pares de valores de todos los campos del formulario. Su nombre coincide con la propiedad cDataField que haya indicado en el punto 4.
- Pares de valores originales de todos los campos del registro. Idéntico nombre al anterior, pero con prefijo 'old_'.
- Clave "append_mode" con valor verdadero si se trata de una operación de alta.

El servicio que se ocupe de la operación de borrado de registros deberá retornar un simple hash con la clave 'pass' a verdadero o falso, según desee.

El ejemplo 25-Tables CRUD le muestra un CRUD completo de una tabla. Si observa con detenimiento verá que no se produce ningún tipo de persistencia en la base datos ubicada en el servidor. Está hecho así aposta para que no se pierdan los datos originales por las pruebas que puedan hacer los usuarios.

La Caché

xaWeb es muy rápido, pero puede serlo aún más si se utilizan técnicas de cache, que en el caso de xaWeb, se pueden realizar a nivel de control. Por ejemplo: un control tipo 'select' que engloba a todos los países del mundo. Cada vez que carga la página web se ha de acceder a una base de datos,

cargar los países e integrarlos en el control. Sabemos que los países no van a cambiar en largo tiempo, por lo tanto, tiene mucho sentido cachear específicamente ese control. xaWeb es capaz de hacer caché no sólo de controles individuales, sino también de controles que engloban a otros controles, como podría ser el pie (footer) de la página web. xaWeb soporta dos tipos de caché:

- A nivel **global**: que se aplica a cualquier solicitud
- A nivel de **sesión**: que se aplica únicamente a la sesión en curso. Es decir, cada sesión tiene sus propios controles cacheados.

Y su puesta en marcha es muy sencilla, sólo tiene que indicar el valor '*global*' o '*session*' a la propiedad oControl:cCache. Eso es todo.

En ambos casos, se distingue la caché por idiomas, es decir, existe una versión de caché por cada idioma que se esté utilizando y se haya designado en la propiedad Document:cLang.

Si se desea anular la caché completamente se recomienda utilizar una única vez el método Document:FlushCache(). Si desea borrar la caché de un determinado objeto, es preferible establecer la propiedad cCache al valor '*flush*'.

NOTA IMPORTANTE

Sólo debe de cachear controles cuyo contenido no cambien absolutamente nada entre llamadas. Incluso la más simple variación impide su uso.

GESTIÓN MULTI-IDIOMA

xaWeb incorpora un paquete de nombre **WTranslator** que es el encargado de realizar esta tarea. Para que un control sea traducido a otro idioma, sólo hace falta que su propiedad *lTranslate* esté a verdadero. En cuanto esto se produce se crea una única instancia de la clase **WTranslator** que será la encargada de realizar todas las traducciones de los controles que tengan la propiedad *lTranslate* a verdadero.

La clase **WTranslator** se apoya en una tabla DBF localizada en *HB_DirBase()* + "***data***" con el mismo nombre que el CGI, y con la siguiente configuración:

- Estructura del fichero: {{ENGLISH},"C", 255, 0}, {SPANISH},"C", 255, 0}}
- Estructura del índice: TAG(1) Name: ENGLISH, Expresión: PADR(ENGLISH, 255)

El gestor interno de traducciones se recupera internamente llamando a la función **Translator**() que retorna un "singleton" del gestor, que por defecto, es una instancia de la clase **WTranslator**. Este gestor de traducciones que incorpora xaWeb, será para muchos usuarios suficiente, pero usted puede realizar su propio gestor si lo desea.

Se ha optado por una tabla DBF para guardar las traducciones porque entendemos que es el sistema más rápido y cómodo para los usuarios de xBase.

Si utiliza servidores Linux para alojar sus páginas es necesario que cree la carpeta **'data'** a partir de la raíz **'usr/lib/cgi-bin**' y otorgue los privilegios de escritura de **grupo** al usuario 'www-data' (Apache). Para ello deberá utilizar las herramientas de Linux **chown** y **chmod** de la siguiente forma:

```
sudo chown :www-data /usr/lib/cgi-bin/data
sudo chmod g+rw  /usr/lib/cgi-bin/data
```

RESALTADO DE CÓDIGO FUENTE

xaWeb incorpora un paquete de nombre **WSyntaxHilite** que permite hacer el resaltado de código **fuente** de forma sencilla. Se apoya en la librería JavaScript HighlighJs y para su uso sólo es necesario instanciar un objeto WSyntaxHilite en el documento principal del CGI: `WSyntaxHilite():New(oDoc)`.

Para utilizarlo sólo hay que insertar el código fuente precedido de las etiquetas <pre> y <code>:

```
<pre><code>Hilite this</code></pre>
```

xaWeb instala una versión especial del CSS de dicha librería para permitir que funcione de forma correcta con el gestor de temas (claro-oscuro) del gestor de contextos. La estructura de la clase WSyntaxHilite permite adaptarla a sus necesidades pudiendo cambiar los colores que utiliza.

Para más información sobre todas las posibilidades de esta librería acceda al siguiente enlace.

xaWeb incorpora un paquete de nombre **WSyntaxHilite** que permite hacer el resaltado de código **fuente** de forma sencilla. Se apoya en la librería JavaScript HighlighJs y para su uso sólo es necesario instanciar un objeto WSyntaxHilite en el documento principal del CGI: `WSyntaxHilite():New(oDoc)`.

Para utilizarlo sólo hay que insertar el código fuente precedido de las etiquetas <pre> y <code>:

```
<pre><code>Hilite this<code></pre>
```

xaWeb instala una versión especial del CSS de dicha librería para permitir que funcione de forma correcta con el gestor de temas (claro-oscuro) del gestor de contextos. La estructura de la clase WSyntaxHilite permite adaptarla a sus necesidades pudiendo cambiar los colores que utiliza.

Para más información sobre todas las posibilidades de esta librería acceda al siguiente enlace.

COMANDO FILE <CFILE> INTO <CVAR>

En los ejemplos podrá comprobar que utilizamos con bastante frecuencia este comando. Este es el mecanismo que podemos utilizar para sustituir los recursos que utilizamos en aplicaciones de escritorio. Es posible añadir cualquier tipo de archivo, incluso binarios, pero si desea incluirlo dentro

de algún control deberá utilizar la función **HB_Base64Encode()** para su conversión. Y a nivel de JavaScript deberá utilizar la función **xw_b64toUnicode().**

EL COMANDO ECHO Y SO

El comando **ECHO** 'texto' que incluye xaWeb simplemente convierte el comando al siguiente código:

```
WText():New( :__WithObject() ):cText := 'texto'
```

Que crea un objeto **WText** y le asigna su propiedad cText. Eso es todo. Observe el constructor de **WText** recibe un valor :WithObject(), que se refiere al objeto que está dentro de un bloque **WITH OBJECT .. END OBJECT**. Por dicho motivo, si utiliza el comando **ECHO** fuera de un bloque **WITH OBJECT .. END OBJECT**, recibirá un error de compilación. En dicho caso podrá indicar directamente el objeto oParent del constructor con el comando: **ECHO** 'texto' **INTO** oHtmlElement.

El comando '**SO**' es los mismo que escribir :__WithObject() y pretende ser el acrónimo de STACK OBJECT.

WSL

Animamos (de forma vehemente ;-) a utilizar WSL (Subsistema de Linux para Windows) con Ubuntu para la creación de los ejecutables en entornos Linux y su posterior ejecución desde un navegador Apache (corriendo también sobre WSL). La nueva versión 9 de Xailer está preparada para que todo el proceso de creación y arranque del navegador con el CGI se realice de forma automática. Más información en nuestro wiki en https://wiki.xailer.com/doku.php?id=linux.

Recuerde que es necesario indicar en el fichero de salida el path completo de WSL donde se creará el CGI, que es: \usrl\lib\cgi-bin\{output-file}

Nota: En el momento que se escriben estas líneas (19-09-24) el navegador integrado de Xailer no soporta operaciones tipo FETCH application/javascript, ni visualización del código fuente de la página y si se invoca el modo desarrollo en el navegador se generan procesos en segundo plano que consumen toda la CPU. Por todos estos motivos se recomienda la utilización de WSL que ofrece un entorno de trabajo tan amigable como puede ser el navegador integrado de Xailer.

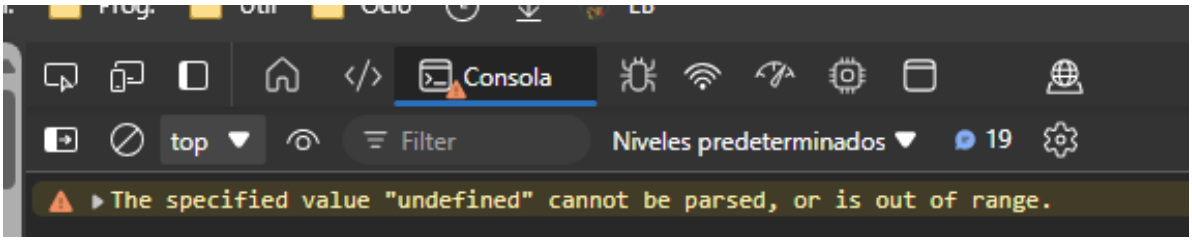
EL LENGUAJE JAVASCRIPT Y HTML

No es un lenguaje difícil de entender para un desarrollador de xBase, pero hay que reconocer que en ocasiones puede ser muy frustrante. Lo más importante que debe de saber con respecto a Harbour, es que se distingue entre mayúsculas y minúsculas en casi todo y por lo tanto es muy importante que se fije en ese aspecto. No es lo mismo un ID de un control con un valor de "First", que un valor de

“first”. Mi recomendación es que utilice siempre minúsculas para todo lo que tenga que ver con establecimiento de propiedades de controles. Le aseguro que se evitará un montón de problemas.

No es necesario saber JavaScript para utilizar xaWeb, pero tarde o temprano le perderá el miedo y empezará a realizar pequeñas funciones para evitar, sobre todo, recargas innecesarias de páginas. Le animo a que eche un vistazo al código JavaScript de xaWeb e intente comprender su funcionamiento. A través de cualquier buscador de Internet encontrará multitud de información de cómo hacer casi cualquier cosa con JavaScript.

El código que realice la fallará sin duda al principio, pero todos los navegadores incluyen potentísimas herramientas de programación que no deberían intimidar a ningún programador de Harbour. Simplemente pulse F12 en el navegador, vaya a la consola y compruebe el error. Es fácil poner puntos de interrupción e ir paso a paso para saber dónde se produjo el error. Mucho ánimo.



Es probable que a pesar de no programar ni una sola línea en JavaScript, no tenga más remedio que acudir a la herramienta de depuración para consultar en la consola si se ha producido un error, ya que observa que su programa no actúa de la forma esperada. Por ejemplo, cuando un campo de un formulario no recoge el dato de un registro de tabla. En la mayoría de los casos se tratará de un simple problema de identificador no encontrado o algo parecido.

NOTA IMPORTANTE

Cuando hace cambios en un módulo JavaScript, casi siempre es necesario provocar un refresco de la página pulsando May+F5.

FUNCIONES JAVASCRIPT DE XAWEB

xaWeb aporta dos mecanismos más de comunicación desde JavaScript con el CGI. El primero de ellos, es utilizando la función JavaScript ***xw_setHbData***. Esta función envía información adicional en la próxima petición tipo ‘**Action**’ o ‘**Form**’. Recuerde que el valor de todas las propiedades de controles que tienen la cláusula ‘**persistent**’ son transmitidas al CGI cada vez que se ejecuta el CGI. Esta función permite añadir cualquier dato adicional de cualquier elemento de la página web. Esta función recibe los siguientes parámetros:

- Elemento HTML (objeto)
- Nombre de la DATA que se desea establecer el valor. Por ejemplo: “cValue”
- Valor
- Si verdadero el envío del dato se hará por método POST en vez de por cookies. Por defecto falso

xaWeb utiliza el mecanismo de Cookies para el trasiego constante de información entre la App CGI y la página web. Lamentablemente las cookies tienen un límite de tan sólo 4 Kbytes por dominio. Por dicho motivo, la función anterior tiene la opción de enviar el dato a través de una operación HTTP POST que prácticamente no tiene límite y se recibe de igual forma en la App CGI. Se ha compatibilizado el envío de datos con esta función y el envío de formularios por método POST.

El segundo mecanismo es de ejecución instantánea a través de un proceso **Fetch**, que se reduce a la llamada de una simple función JavaScript de nombre **xw_GetHbData**. Como su nombre indica, esta función recupera cualquier dato de su CGI de forma directa, **sin que se abandone la página web**. El valor retornado por la función ha de ser tratado desde nuestra página web y dependiendo de cómo haya establecido el sistema de recuperación en su segundo parámetro, recibirá un JSON que deberá procesar o recibirá código JavaScript que se autoejecutará en su página web. Estos son los parámetros de la función:

- Nombre del método en nuestro documento **WDoc** del CGI.
- Nombre del propio documento (Wdoc:cName). Opcional, por defecto el documento actual.
- Tipo de operación: “application/json;charset=utf-8” o "application/javascript". Opcional, por defecto, el primero.

Los Ejemplos

Los ejemplos han sido numerados para que sean probados y analizados en el mismo orden que se indica. Arrancan de un sencillo ‘Hola mundo’ y se van complicando poco a poco, intentando mostrar todas las posibilidades de xaWeb.

Los ejemplos no pretenden ser espectaculares en cuanto a vistosidad e incluso, los primeros no utilizan ningún tipo de Framework, precisamente para mostrar la esencia del código, tanto a nivel PRG como HTML.

Puede ver todos los ejemplos corriendo sobre un servidor Linux en: All together

La Licencia

xaWeb es un producto comercial, propiedad de OZ Software. No obstante, la versión de demostración gratuita que incluye el 90% de las fuentes de programación es completamente operativa y es factible realizar con ella pequeños proyectos. Cuando el proyecto se hace más grande, es necesario adquirir la licencia comercial.

El precio de la versión comercial aún no ha sido asignado, pero en cualquier caso será un precio muy ajustado que esperamos esté alrededor de los 150 euros y existirá una oferta de lanzamiento con un importante descuento.

La licencia es perpetua y se acompaña con una suscripción anual que incluye:

- Actualizaciones de xaWeb durante doces meses desde su compra
- Soporte técnico en nuestros foros
- Acceso a un curso on-line de introducción a xaWeb de 8 horas
- La no renovación antes de termine la suscripción anual implica que deberá volver a comprar el producto completo si desea acceder a las nuevas versiones de xaWeb

La renovación de la suscripción tiene un coste del 50% del valor de la licencia comercial en el momento en que se realice la renovación.

Es posible que, en el futuro, existan distintas versiones de xaWeb, dependiendo de la funcionalidad que ofrezcan.

El Despliegue

El despliegue consiste en copiar en las carpetas situadas en la nube, donde se alojarán las páginas web, los ejecutables y cualquier otro fichero necesario para que nuestra aplicación web funcione correctamente. Este trabajo se puede realizar manualmente a través de un cliente FTP cualquiera. No obstante, el IDE de Xailer tiene un '**plugin'** específico para ello realizado por mí mismo y que puede obtener toda la información sobre el mismo en este enlace.

Colaboración

Existe un grupo reducido de alfa/beta testers que reciben de forma periódica las mejoras y corrección de errores que se realicen en xaWeb. Durante esta fase se entregarán todas las fuentes de xaWeb excepto un par de módulos por motivos de salvaguarda de la propiedad intelectual.

Nos interesa conocer, sobre todo, los procesos clásicos que los usuarios requieren en su software de gestión Web, para intentar simplificarlos al máximo, discutiendo en el canal la mejor forma de hacerlo.

Si desea **colaborar** con xaWeb le animamos a que se una al grupo alfa/beta testers mandando un correo a la dirección iozuniga@ozs.es solicitándolo. Recalcamos la palabra 'colaborar', porque será

NOTA IMPORTANTE

Si va a utilizar el IDE de Xailer para crear sus CGIs de xaWeb necesitará Xailer 9.1.

necesario que su colaboración sea activa. En caso contrario, nos reservamos al derecho a sacarle de la lista de beta-testers.

AGRADECIMIENTOS

Quisiera agradecer a:

- Domenic Corso
- Kevin Powell

Los vídeos de HTML, CSS y JavaScript de estos autores han sido de gran inspiración. Importantes porciones de su código han sido incorporadas a xaWeb con los oportunos reconocimientos de copyright.