



TWeb for modHarbour 

Preámbulo	3
Funcionamiento del sistema de rejillas.....	5
Configuración de Tweb.....	7
Inicialización de TWeb. Carga de librerías	8
Row vs RowGroup	11
Html / Html inline.....	13
Rows Valign/Halign.....	17
Sistema grid responsive.....	20
Integración de javascript.....	24
Ejecutar Formulario	25
Via Form	25
Via Ajax	26
Controles.....	29
Clausulas Comunes	29
GET	30
FONT.....	30
RADIO, SELECT	30
FOLDER.....	31
BROWSE / GRID.....	31
Otros controles	36
Complementos de interés	36
Get Autocomplete	36
Upload Files	38
Templates → Html File.....	39
Utilidades Javascript	44
Charset.....	45
Charset – Dbf.....	47
Conclusiones.....	48

Preámbulo

Cuando uno entra en el mundo Web empieza a ver, estudiar, probar numerosos entornos que permiten crear páginas web de mil maneras diferentes, eso sí con una curva de aprendizaje muy alta y un periodo de adaptación a la nueva manera de programar, más o menos largo.

Los que venimos de entornos de programación (especialmente Windows), nos hemos curtidos en mil batallas hasta aprender, asimilar y sentirnos cómodos con este sistema operativo. En muchos lenguajes, con los años hemos aprendido las clases a fondo de nuestro lenguaje de programación y nos ha permitido crear nuestras aplicaciones de gestión, y subrayo gestión porque básicamente es el modelo que más usamos en este entorno. Así pues, si quiero dar el salto a la web, ¿necesito ser un experto en todos los lenguajes que tendré de usar para poder generar una aplicación típica de mantenimiento y gestión ?

Este ha sido el objetivo de crear esta librería, el poder facilitar el salto a la Web de una manera fácil usando nuestra filosofía de trabajo actual. No pretendo con esta librería inventar la rueda, ni hacer magia pura, simplemente que su uso nos facilite la entrada poco a poco a la Web y perder el miedo ha desarrollar nuestras aplicaciones y especialmente nuestras pantallas, ofreciéndonos una transición más fácil y suave a todo este entorno. Muchos de los que usamos librerías de programación como [FWH](#) para Windows (la mejor librería de programación para win de Antonio Linares), al principio no conocíamos las clases, ni siquiera las entendíamos. Muchas estaban escritas en C, pero allí estaban, las usábamos y funcionaban.

Con el tiempo las hemos modificado, cambiado, ... e incluso hemos desarrollado de nuevas. Pues ahora se trata de lo mismo. Usar estas clases, sentirnos cómodos y si nos vemos capaces, ir aumentando las diferentes funcionalidades. Entrar sin miedo en este nuevo entorno y empezar a diseñar estas pantallas Web con nuestra metodología que hemos aprendido y asimilado durante años al estilo xBase, con nuestros queridos comandos. TWeb se ha basado en esta librería madurando mas el concepto hasta portarlo a la Web.

Inicialmente esta librería la cree para php, usando una serie de clases que me permitía diseñar muy fácilmente y rápidamente estas pantallas, incluso se diseño una manera para diseñarlas usando un editor de recursos de Windows. El sistema funcionaba perfectamente, pero con el tiempo nos dimos también cuenta que estas pantallas "estáticas" estilo Windows quedaban algo obsoletas con el estilo de pantallas web que últimamente se pueden ver en la web usando estilos responsive, en el cual puedes observar que todos los elementos de la pantallas son dinámicos, se mueven, adaptan, fluyen...

Con la entrada de modharbour cree la librería Mercury que te permite estructurar perfectamente un programa dentro de este entorno web usando arquitectura MVC. El sistema esta muy estabilizado y va muy bien, pero nos quedaba una pata por acabar de definir y son las vistas, como poder trabajarlas

Cuando creabas las vistas (views) las realizamos en code html/css puro y duro y aquí es donde mucho de nuestros programadores hacían el alto. Por esto he creado esta versión Tweb para modharbour usando toda la potencia

del preprocesador (algo muy añorado en php) que nos ayudara de una manera brutal al diseño de estas pantallas, trabajando de una manera muy amigable.

En esta primera versión estarán disponibles los controles mas básicos pero esta diseñado para poderlo escalar fácilmente a nuevos controles y funcionalidades.

TWeb se puede usar solo o juntamente con Mercury y el objetivo principal es la del diseño de las pantallas. Para esta versión se usara el framework Bootstrap uno de los mas populares actualmente en el mundo web. Ser un maestro en el "maquetado" de la web lleva muchos años de aprendizaje, Bootstrap sintetiza muchos aspectos, Tweb es una capa que lo intenta facilitar aun mas.

No pretendo y ni es la idea enseñar html, javascript ni css, por lo que como mínimo el usuario debe conocer a un nivel muy básico el lenguaje. Intentaremos poder trabajar de una manera fácil, ágil y productiva.

TWeb es el resultado de muchas horas de investigación y de ver como encajan mejor las piezas para poder disponer de este sistema.

Cualquiera está invitado a sugerir, aportar y aumentar la funcionalidad y potencia del sistema con nuevas ideas, tips, clases, conceptos,... Intento crear una base para poder empezar a trabajar abierta a todo el mundo.

Siempre he creído que la web es de todos y nosotros debemos decidir como poderla manejar, solo quiero ofrecer ahora a toda la comunidad harbour "otro" estilo y manera de poder realizar nuestros sueños en la programación web. Lo conseguiremos 😊

Un saludo a todos y que empiece la fiesta...

Funcionamiento del sistema de rejillas

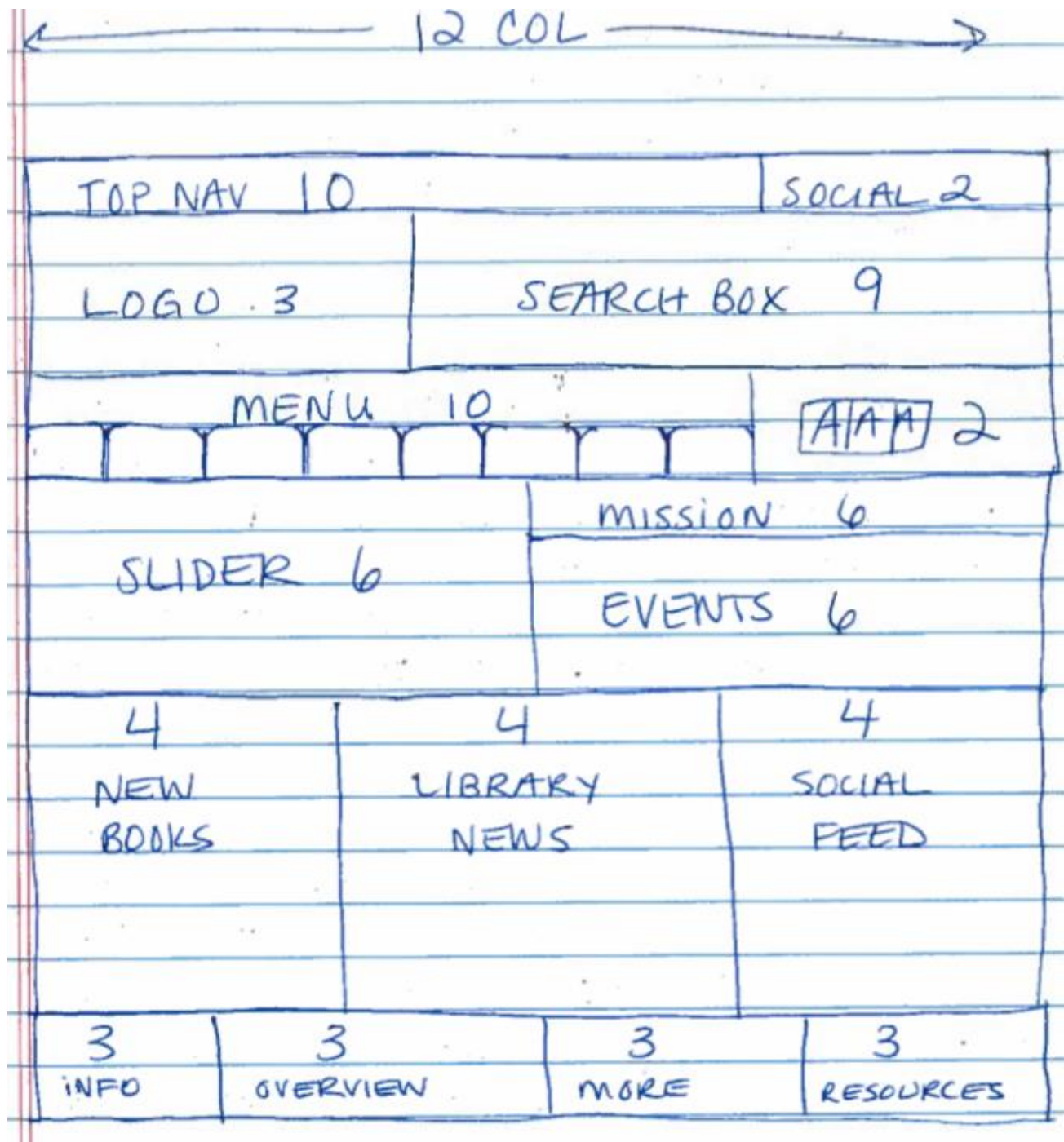
Bootstrap utiliza un sistema de rejillas para poder diseñar las pantallas. Si bien es cierto que Tweb esta diseñado para utilizar Bootstrap, no es especialmente obligatorio. Hay mucha documentación acerca de como funciona a nivel muy detallado, pero básicamente imaginarnos una pantalla formada por líneas y 12 columnas. En cada línea podemos ponemos nuestros controles que ocuparan tantas columnas como sea necesario de estas 12. Podemos por ejemplo crear una línea (row) y esta línea dividirla en 3 partes. Una estará formada por 6 columnas, otra por 4 y otra por 2 columnas.

No es necesario utilizar las 12, pero estas 12 forman todo el ancho de la columna. Cada columna será como un nuevo espacio, que podremos insertar mas líneas y 12 columnas...

Será el sistema Bootstrap quien se encargará de distribuir en estos espacios los diferentes controles, adaptándolos a las dimensiones de cada pantalla. Bootstrap tiene muchas funcionalidades para poder adaptar de muchas maneras nuestras pantallas, y conocerlo a fondo es tarea de mucho estudio. Tweb intenta minimizar este impacto inicial, ayudándote de una manera inicial a crear rápidamente estas pantallas. Con el tiempo se ira asimilando estas técnicas y se podrán combinar las funcionalidades de Tweb con código nativo html/css puro y duro.



Llevado a la práctica y mirando de crear una pantalla a lápiz y papel (mockup) podríamos crear un ejemplo para entender como se encajan las diferentes partes en esta rejilla



Es decir, a partir de ahora habremos de pensar en clave 12 columnas, se acabó pensar en posiciones fijas de la pantalla 😊

Configuración de Tweb

TWeb lo podrás bajar desde el siguiente repositorio → <https://github.com/carles9000/tweb>

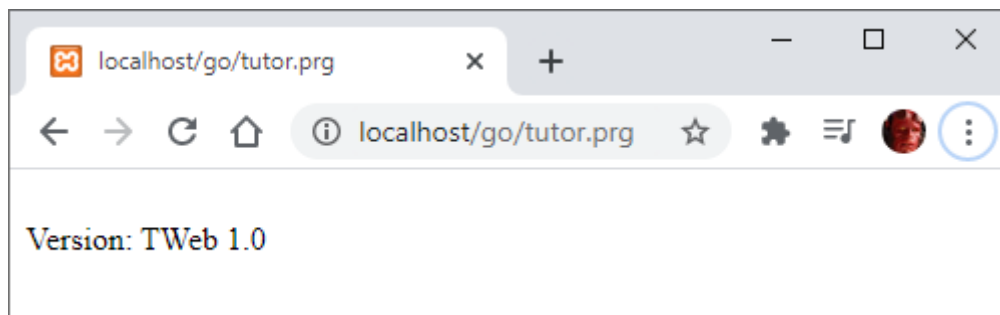
Por defecto instalaremos la librería dentro de la carpeta lib de nuestra carpeta de proyecto, p.e. si tenemos una carpeta de proyecto que llamamos go, la estructura quedaría así



Partimos de la base que ya tenemos modharbour instalado y funcionando correctamente. Lo primero que debemos hacer es crear un pequeño prg para comprobar que cargamos bien la librería. Podemos revisar el código tutor.prg en el que se muestra como crear este test inicial.

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}  
  
function main()  
  
    ? 'Version:', twebversion()  
  
retu nil
```

Básicamente y como sabias cargamos la librería y ejecutamos la función twebversion(). Si se carga todo nos debería aparecer una pantalla como esta



En este caso ya tenemos todo preparado para trabajar con Tweb 😊

Inicialización de TWeb. Carga de librerías

Lo primero que debemos de entender es que Tweb se basa en Bootstrap, por lo que debemos de cargar sus librerías así como otras que usamos en la propia Tweb. Es por esto que lo primero que debemos de hacer es inicializar nuestro programa y cargar estas librerías.

tutor1.prg

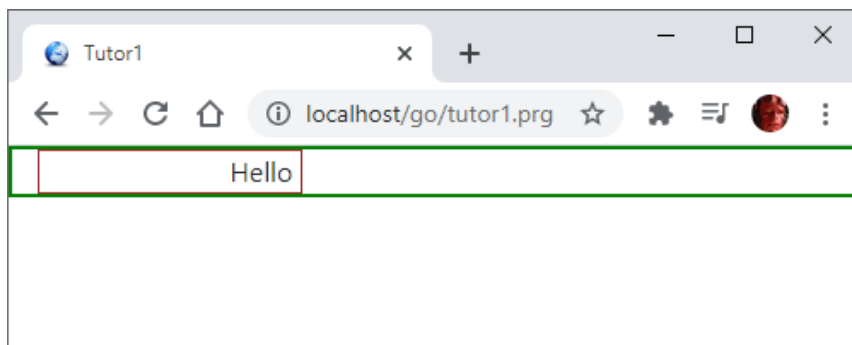
<pre>// {% LoadHrb('lib/tweb/tweb.hrb') %} #include {% TWebInclude() %} function main() LOCAL oWeb, o DEFINE WEB oWeb TITLE 'Tutor1' INIT DEFINE FORM o o:lDessign := .T. INIT FORM o ROW o SAY VALUE 'Hello' OF o END o END FORM o retu nil</pre>	<p>Carga de la librería Tweb</p> <p>Incluimos fichero de preprocesado de comandos de Tweb</p> <p>Definimos una web → Este proceso carga las librerías necesarias</p> <p>Definimos un formulario Assignamos .T. a la propiedad lDessign</p> <p>Inicializamos el formulario</p> <p>Creamos una linea Pintamos con un SAY "Hello" Finalizamos una linea</p> <p>Fin de formulario</p>
---	---

Esto es la base de Tweb -> Inicializar el sistema y pintar controles !!!

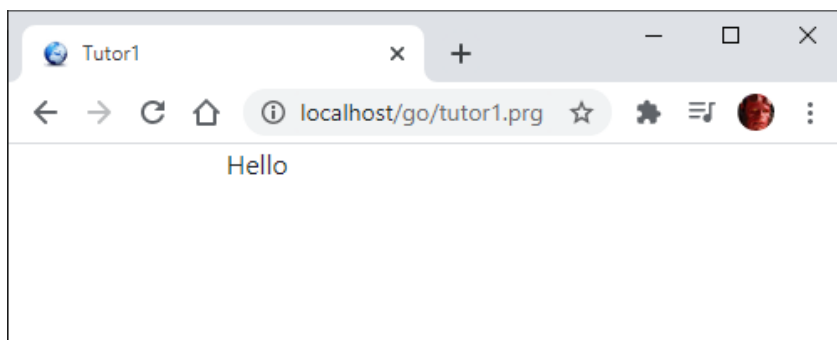
El objetivo principal es aprender a pintar la pantalla responsive. Otro tema muy importante y que vamos a seguir en este manual es el concepto de las 12 columnas de Bootstrap y por donde dibujamos nosotros nuestros controles. Es por esta razón que cuando definimos nuestro formulario podemos especificar la propiedad → lDessign = .T. que nos pintará los bordes de todo lo que especifiquemos para ver (y aprender) por donde se mueve el sistema Bootstrap 😊

A medida que vayamos poniendo controles veremos un montón de líneas (siempre que tengamos lDessign == .T.) y esto nos ayudará a entender como funciona el pintado (que es muy complejo en muchos casos).

Este código simple, inicializa el sistema, carga las libs, y pintamos un control say, todo en modo diseño. El resultado seria el siguiente



Como se puede observar iremos marcando las líneas para aprender como el sistema dibuja todo. Si ejecutáramos con la IDesign := .F. podemos observar la diferencia de pintado



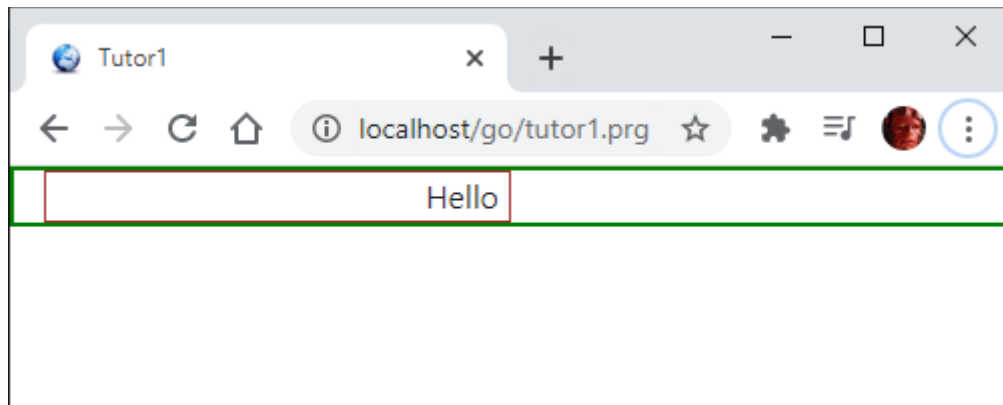
Podemos observar que realmente nos pinta, pero si no podemos entender porque en esa posición no aprenderemos como realmente actúa Bootstrap

Continuando con este primer ejemplo podemos observar una línea verde que nos marca la delimitación del formulario (DEFINE FORM) y la delimitación del objeto say (SAY value "Hello"). Por defecto los controles ocuparan 4 columnas y es lo que esta marcando el recuadro rojo que vendrá a ser 1/3 parte del ancho.

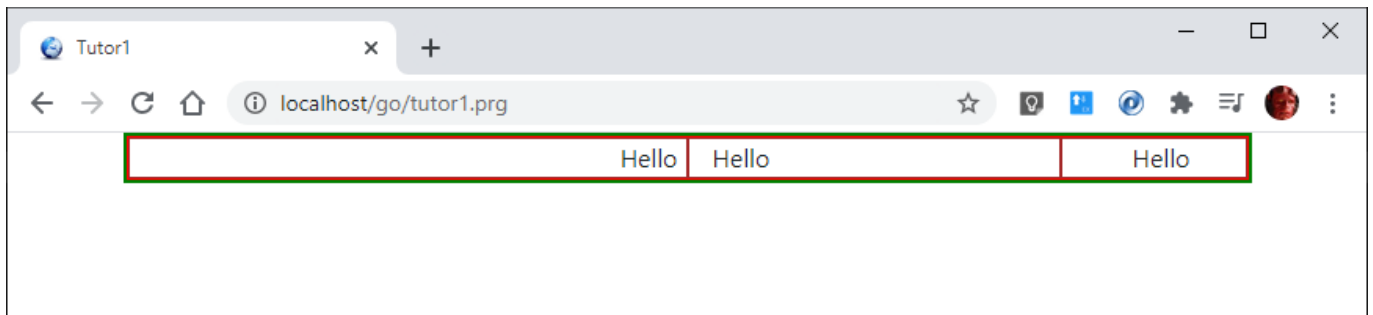
A medida que vayamos avanzando iremos aprendiendo el manejo de los comandos xBase y el posicionamiento de los diferentes controles. Por ejemplo, cada control tiene una clausula GRID <nCols> que indicamos cuanto queremos que nos ocupe el control en la línea (recordad que tenemos 12 columnas). Si p.e ponemos en este ejemplo

```
SAY VALUE 'Hello' GRID 6 OF 0
```

El control, ocuparía la mitad de la línea de la pantalla

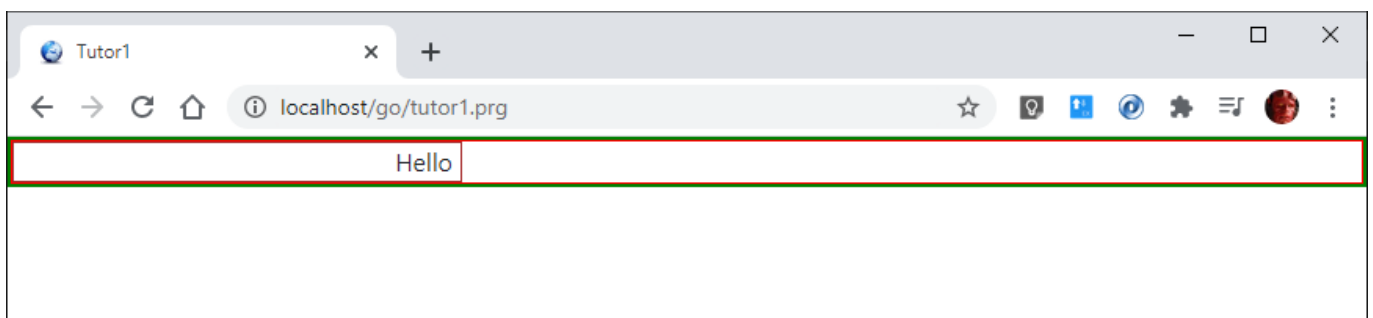


Imaginemos para acabar en este ejemplo que queremos poner 3 controles SAY que ocupen cada uno de ellos p.e 6,4 y 2 columnas, alineados derecha, izquierda, centro



Para finalizar este capítulo, el control FORM tiene otra propiedad que llamaremos → IFluid, por defecto == .F.

Cuando este a .T. adaptará el pintado a todo el ancho del dispositivo, sino form deja unos márgenes a la derecha e izquierda



A medida que probéis los ejemplos es importante ir redimensionado el browse (navegador) para poder asimilar como se comporta el sistema

tutor2.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}  
  
#include {% TWebInclude() %}  
  
function main()  
  
    LOCAL o  
  
    DEFINE WEB oWeb TITLE 'Tutor1' INIT  
  
    DEFINE FORM o ID 'demo'  
        o:lDessign := .T.  
        o:lFluid   := .T.  
  
    INIT FORM o  
  
        ROW o  
            SAY VALUE 'Hello' GRID 6 ALIGN 'right' OF o  
            SAY VALUE 'Hello' GRID 4 OF o  
            SAY VALUE 'Hello' GRID 2 ALIGN 'center' OF o  
        END o  
  
    END FORM o  
  
retu nil
```

Row vs RowGroup

Hemos visto que el comando nos crea una línea nueva. Podemos crear tantas líneas como queramos, pero el comando ROW nos crea una línea pegada a la siguiente, mientras que si usamos ROWGROUP deja un pequeño margen en la parte inferior. Vemos la diferencia de usar uno u otro con este code.

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}  
  
#include {% TWebInclude() %}  
  
function main()  
  
    LOCAL o, oWeb  
  
    DEFINE WEB oWeb TITLE 'Tutor3' INIT  
  
    DEFINE FORM o ID 'demo'  
        o:lDessign := .T.  
  
    INIT FORM o
```

```

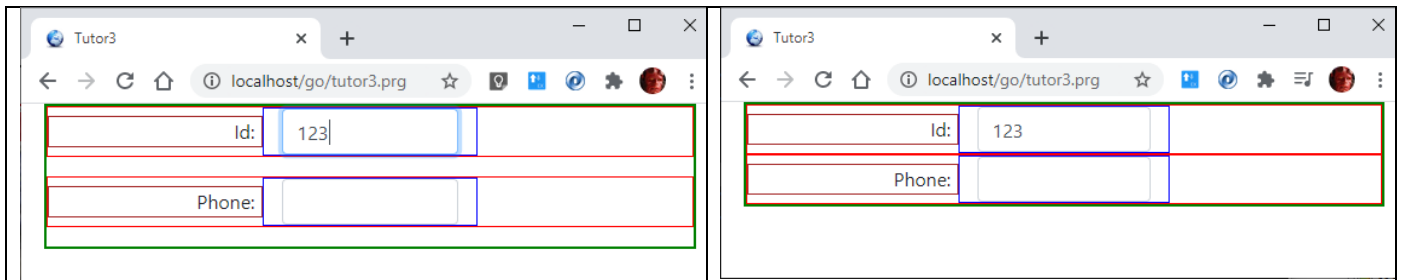
    ROWGROUP ○
        SAY VALUE 'Id:' ALIGN 'right' OF ○
        GET VALUE '123' OF ○
    END ○

    ROWGROUP ○
        SAY VALUE 'Phone:' ALIGN 'right' OF ○
        GET VALUE '' OF ○
    END ○

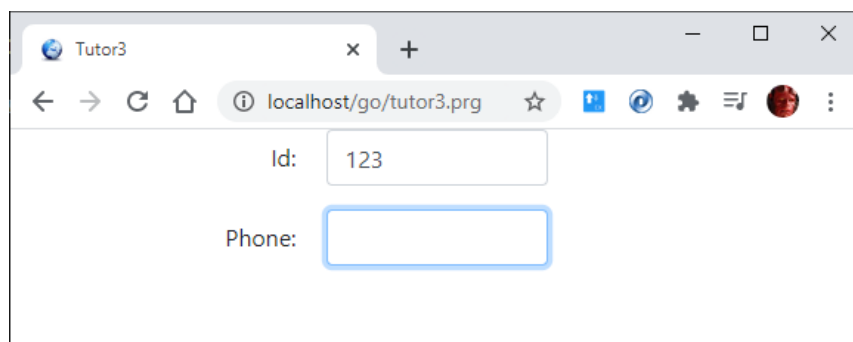
END FORM ○

retu nil

```



Recordad que iremos mostrando las pantallas en este tutorial de maquetado con Tweb en modo IDesign. En cualquier momento podéis cambiar la propiedad IDesign a .F. para poder comprobar como quedaría realmente.



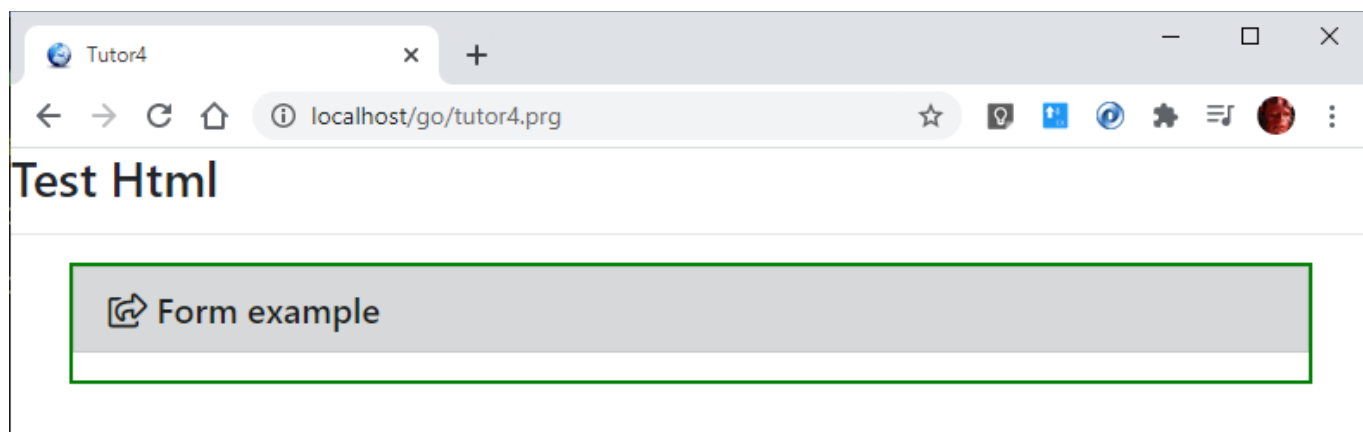
Html / Html inline

TWeb nos permite combinar nuestro código xBase con código normal y fácilmente podemos combinarlo usando el comando HTML. Tenemos 2 variantes:

- Usar en una misma línea un pequeño código → HTML o INLINE '`<h3>Test Form</h3><hr>`'
- Insertar todo el código que queramos y al finalizar poner el comando ENDTEXT

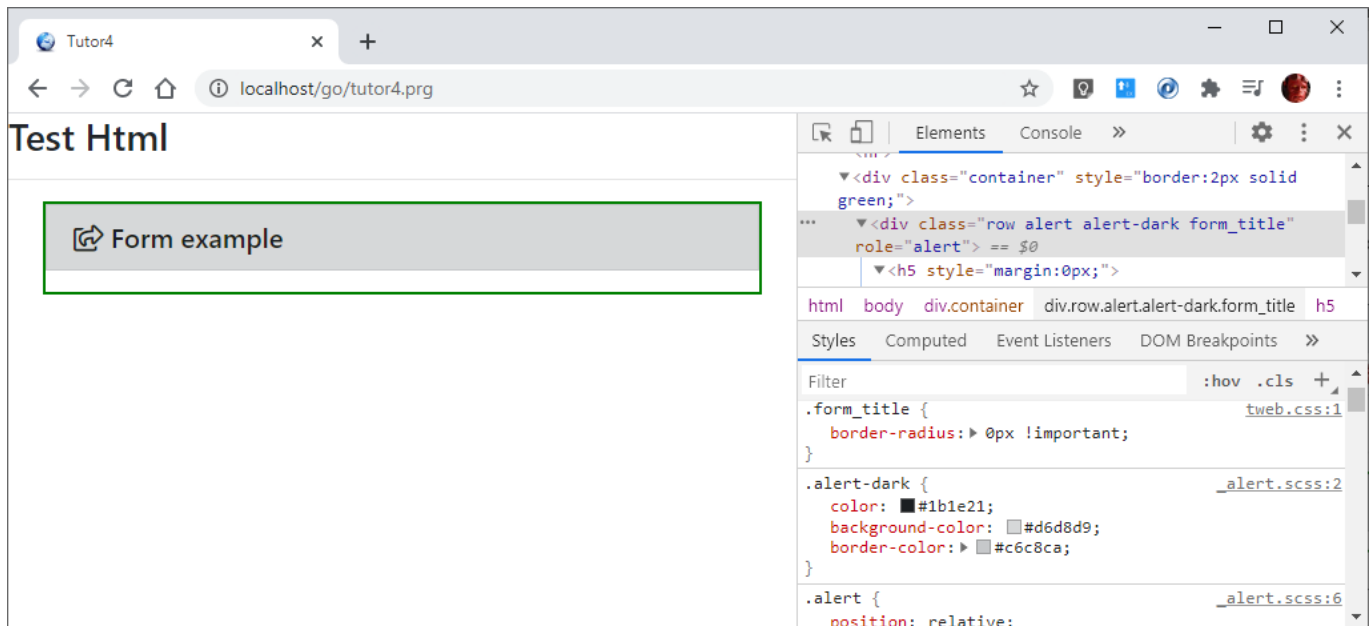
```
HTML o
  <div class="row alert alert-dark form_title" role="alert">
    <h5 style="margin:0px;">
      <i class="far fa-share-square"></i>
      Form example
    </h5>
  </div>
ENDTEXT
```

De esta manera fácilmente podemos poner el código html/css/javascript dentro de nuestro diseño , dándonos el control total sobre la página



Una de las ventajas de usar Tweb es que es una librería independiente para harbour que no necesita el uso de otros programas para generar nuestras páginas, multiplataforma (windows/Linux) y no necesita compilaciones. Escribir código y ejecutar, no hay mas. Otros frameworks muestran un abanico de opciones para poder ver las propiedades de cada elemento del DOM, pero creo que la mejor manera es usar el propio inspector que lleva el propio navegador para ir probando, modificando,... y luego aplicar los cambios directamente en nuestro código.

Quizás es un nivel mas pero a medida que vayáis avanzando seguro usareis el inspector (por que además es vital), pero de esto ya se hablará en otro momento...



En este punto del manual ya debemos entender como iremos definiendo nuestras pantallas. Este pequeño ejemplo podemos observar como vamos mezclando los diferentes conceptos, añadiendo rows, controles, código html puro y duro,...

tutor5.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

  local o, oWeb
  local cLoren := "<h2>Why do we use it?</h2>It is a long established fact that a reader will be ..."

  DEFINE WEB oWeb TITLE 'Tutor5' INIT

  DEFINE FORM o
    o:Idesign      := .t.
    o:IFluid       := .f.

    HTML o INLINE '<h3>Test Form</h3> <hr>'

    INIT FORM o

      HTML o
        <div class="row alert alert-dark form_title" role="alert">
          <h5 style="margin:0px;">
            <i class="far fa-share-square"> </i>
            Form example
          </h5>
        </div>
      ENDTEXT

      ROWGROUP o
        SAY VALUE 'Id:' ALIGN 'right' OF o
        GET VALUE "" OF o
      END o

      ROWGROUP
        SAY VALUE 'Phone:' ALIGN 'right' OF o
        GET VALUE "" PLACEHOLDER "Enter phone number" OF o
      END o

      ROWGROUP o
```

```

BUTTON LABEL ' Test Button' ACTION "alert( 'Hi!' )" GRID 8 ALIGN 'right' ;
ICON '<i class="fas fa-clipboard-check"></i>' ;
CLASS 'btn-danger btnticket' OF o

```

```

END o

```

```

ROWGROUP o

```

```

SAY VALUE cLoren GRID 12 OF o

```

```

END o

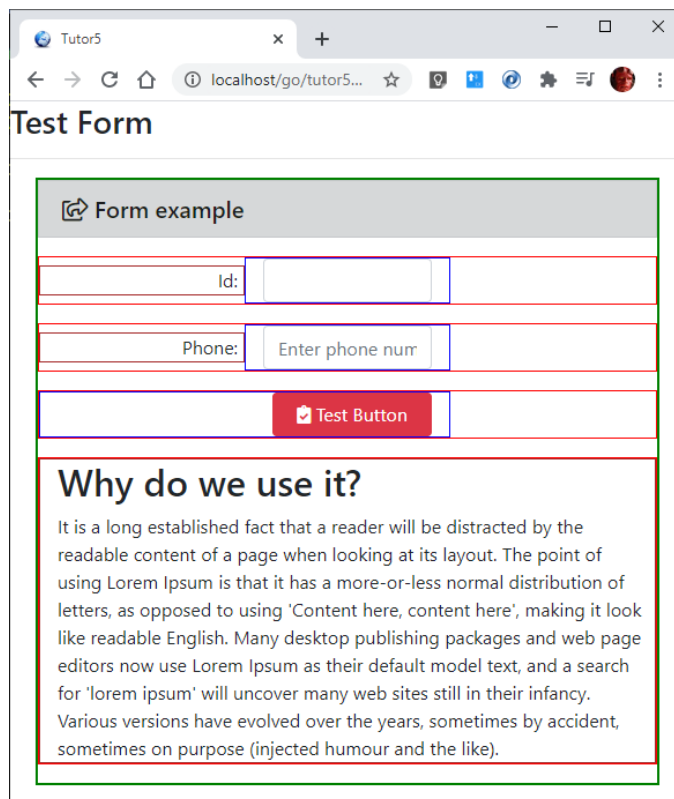
```

```

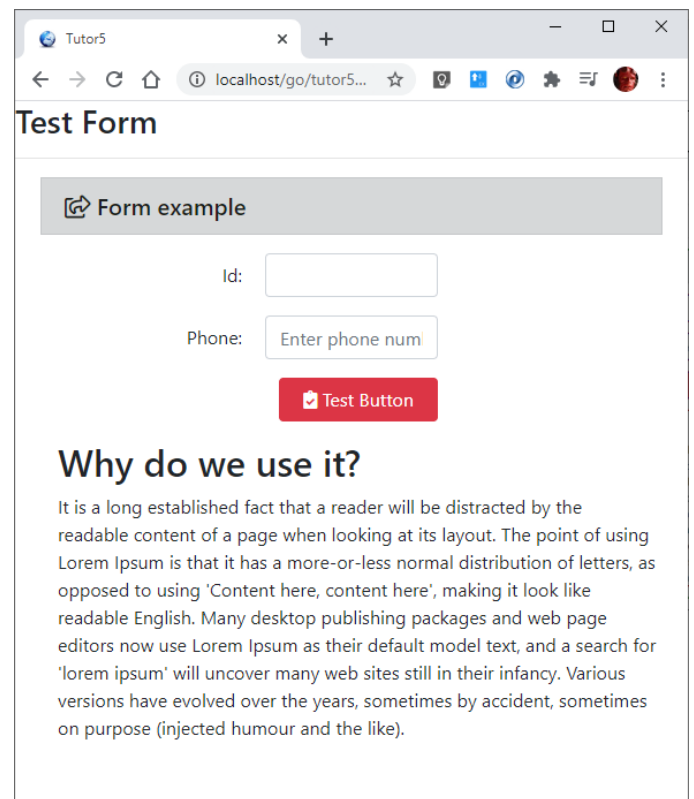
END FORM o

```

retu nil



The screenshot shows a web browser window with the URL 'localhost/go/tutor5...'. The page title is 'Test Form'. Below the title, there is a section titled 'Form example' with a green border. Inside this section, there are three input fields: 'Id:', 'Phone: Enter phone num', and a red button labeled 'Test Button'. Below the form example, there is a text block titled 'Why do we use it?' with a red border. The text block contains a paragraph about Lorem Ipsum and its use in web design.



The screenshot shows a web browser window with the URL 'localhost/go/tutor5...'. The page title is 'Test Form'. Below the title, there is a section titled 'Form example' with a grey background. Inside this section, there are two input fields: 'Id:' and 'Phone: Enter phone num', followed by a red button labeled 'Test Button'. Below the form example, there is a text block titled 'Why do we use it?' with a white background. The text block contains a paragraph about Lorem Ipsum and its use in web design.

Rows Valign/Halign

Cuando creamos una línea (row) podemos poner varios controles, textos, ... y cada uno puede ocupar mas espacio que otro ya sea verticalmente u horizontalmente. Por defecto una row alinea en la parte superior ('top'), pero podemos indicar que se alinea verticalmente en el 'center' o en la parte inferior ('bottom') con la clausula VALIGN <cValue>

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb
    LOCAL aTxt := { 'Volvo', 'Seat', 'Renault' }
    LOCAL aKey := { 'V', 'S', 'R' }

    DEFINE WEB oWeb TITLE 'Tutor 7' INIT

    DEFINE FORM o
        o:lDessign := .T.
        o:cSizing  := 'sm'           // SM/LG

    INIT FORM o

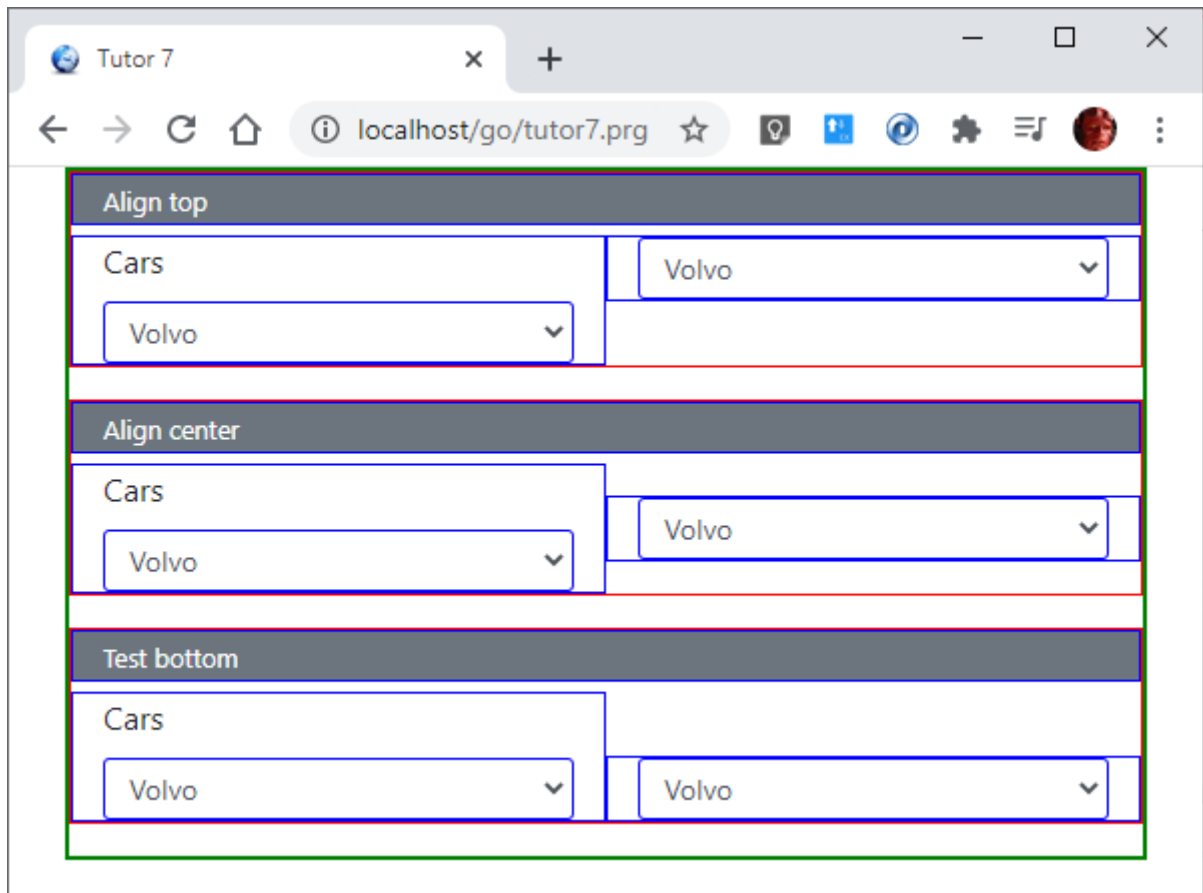
        ROWGROUP o VALIGN 'top'
            SEPARATOR o LABEL 'Align top'
            SELECT oSelect LABEL 'Cars' PROMPT aTxt VALUES aKey GRID 6 OF o
            SELECT oSelect                                PROMPT aTxt VALUES aKey GRID 6 OF o
        END o

        ROWGROUP o VALIGN 'center'           // Default
            SEPARATOR o LABEL 'Align center'
            SELECT oSelect LABEL 'Cars' PROMPT aTxt VALUES aKey GRID 6 OF o
            SELECT oSelect                                PROMPT aTxt VALUES aKey GRID 6 OF o
        END o

        ROWGROUP o VALIGN 'bottom'
            SEPARATOR o LABEL 'Test bottom'
            SELECT oSelect LABEL 'Cars' PROMPT aTxt VALUES aKey GRID 6 OF o
            SELECT oSelect                                PROMPT aTxt VALUES aKey GRID 6 OF o
        END o

    END FORM o

retu nil
```



De la misma manera podemos alinear a nivel horizontal, por defecto se alinea por la izquierda

tutor8.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'Tutor 8' INIT

    DEFINE FORM o
        o:lDessign := .T.
        o:lFluid   := .T.

    INIT FORM o

        ROW o HALIGN 'center'
            SAY VALUE 'Hello' GRID 4 ALIGN 'right' OF o
            SAY VALUE 'Bye'   GRID 2 OF o
```

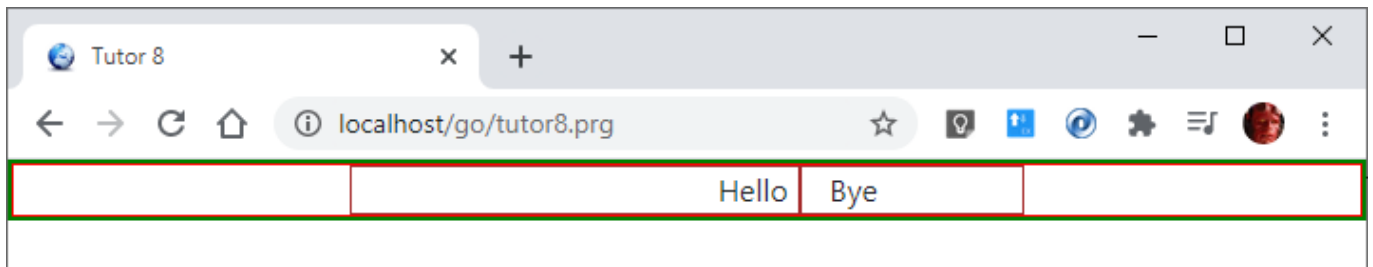
```

END o

END FORM o

retu nil

```



Siempre que definimos un formulario internamente se trataran los controles con un tamaño normal por defecto, per podemos especificar que se traten con un tamaño pequeño o grande. Para ello utilizaremos la clausula cSizing := 'sm' para pequeños (small) o cSizing := 'lg' para grandes (large)

```

//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'Tutor 9' INIT

    DEFINE FORM o
        o:lDessign := .F.
        o:cSizing  := 'lg'           // SM/LG (small/large)

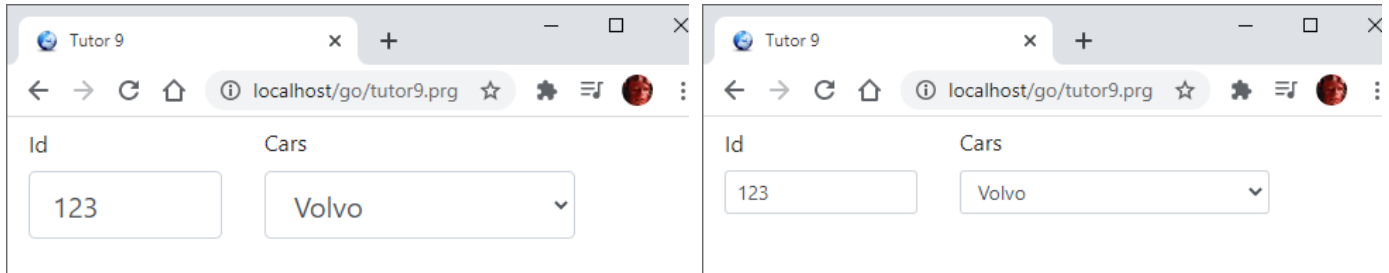
    INIT FORM o

        ROWGROUP o VALIGN 'bottom'
        GET VALUE '123' LABEL 'Id' OF o
        SELECT LABEL 'Cars' PROMPT 'Volvo', 'Renault' VALUES 'V', 'R' GRID 6 OF o
    END o

    END FORM o

retu nil

```



Sistema grid responsive

Hasta ahora hemos visto como hemos de combinar con estos pocos comandos para situar nuestros controles en pantalla y que se adapte bien. Nos queda ver la manera en que podemos controlar (si queremos o necesitamos) las columnas a medida que redimensionamos la pantalla. Para ello usaremos este código para poder entender este capítulo.

Definiremos una row y dentro de ella pondremos 2 columnas con texto cada una de ellas de 6 de ancho. La idea es que se se distribuyen a lo ancho de la pantalla pero queremos controlar a la hora de redimensionar la pantalla cuando la hacemos mas pequeña (o se use esta pantallas en una table o smartphone) que una columna se posicione debajo de la otra para poder ver mejor la información.

tutor10.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o
    LOCAL cTxt := ''

    DEFINE WEB oWeb TITLE 'Tutor8' INIT

    DEFINE FORM o ID 'demo'
        o:lDesign      := .F.
        o:cType        := 'lg'      //  xs,sm,md,lg
        o:cSizing      := 'sm'      //  sm,lg

    HTML o
        <div class="alert alert-dark form_title" role="alert">
            <h5 style="margin:0px;">
                <i class="far fa-share-square"></i>
```

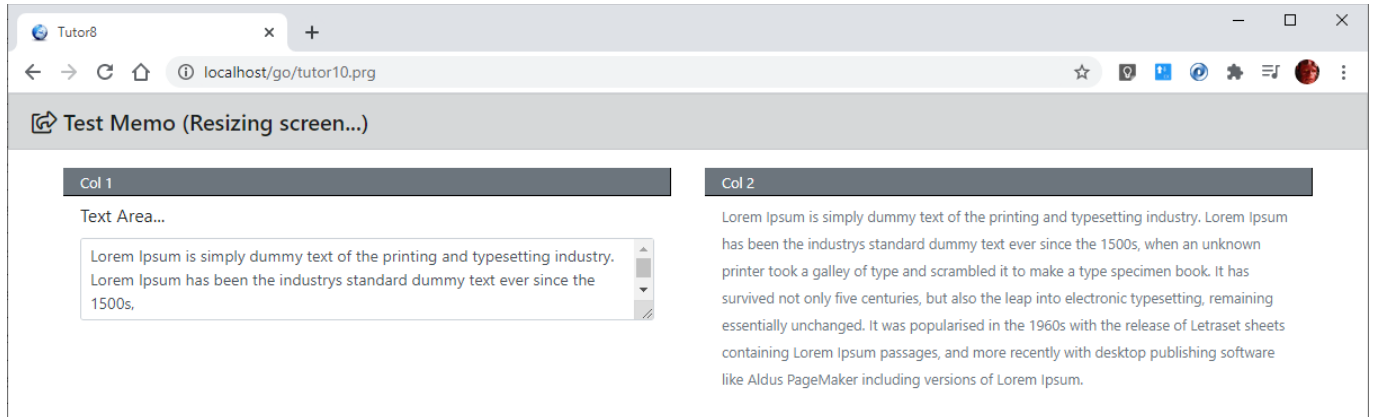
```
Test Memo (Resizing screen...)
    </h5>
</div>
ENDTEXT

TEXT TO cTxt
Lorem Ipsum is simply dummy text ...
ENDTEXT

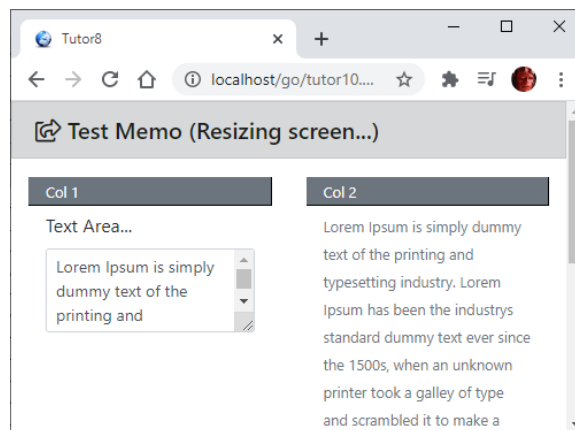
INIT FORM o
    ROW o VALIGN 'top'
        COL o GRID 6
            SEPARATOR o LABEL 'Col 1'
            GET oGet MEMO LABEL 'Text Area...' VALUE cTxt GRID 12 OF o
        END o
        COL o GRID 6
            SEPARATOR o LABEL 'Col 2'
            SMALL o LABEL cTxt GRID 12
        END o
    ENDROW o
END FORM o

retu nil
```

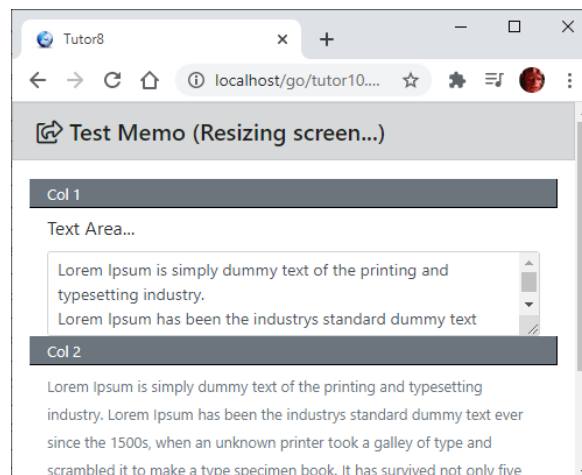
Esta pantalla vista en pantalla de escritorio tendría una apariencia similar a esta



Por defecto a medida que la hacemos mas estrecha mantiene la definición que hemos diseñado



Pero podemos controlar el flujo para decir que a partir de un ancho, cabalgue y se ponga una columna encima de la otra



Este punto final de control lo haremos con la propiedad `cType` del objeto `Form`, y podrá tener los siguiente valores: `xs`, `sm`, `md`, `lg` (`extra small`, `small`, `médium`, `large`)

En función del valor el sistema recolocará los controles de una manera u otra. Se trata de ir probando con estos 4 valores e ir redimensionando el tamaño del navegador para ver y entender como funciona le sistema. Por defecto Tweb no tiene valor, por lo que el sistema mantendrá nuestras columnas como las hemos definido. Evidentemente esto no significa que se vea bien la pantalla cuando esta se vea p.e desde un `smartphone`, por lo que para cada caso de pantalla tendrá un diseño u otro.

De esta manera tan fácil podremos controlar el flujo de las diferentes partes de nuestra pantalla para que tenga un comportamiento `responsive`

Bootstrap ofrece muchas maneras de ejercer este tipo de control y dentro de su complejidad inicial te permite junto a la manipulación del `css` diseñar pantallas que se adapten perfectamente en función de dispositivo y contenido. Podéis leer mas sobre el tema en este enlace → <https://getbootstrap.com/docs/4.0/layout/grid/>

Y este es objetivo como hemos dicho de Tweb, ayudar a crear rápidamente y sobre todo fácilmente pantallas `responsive` de aspecto profesional

The screenshot shows a web browser window with the address bar displaying `localhost/go/spagetti.prg`. The page has a header image of a city skyline with the text "Friends of mod harbour" in blue. Below the header, there is a section titled "Contact Us" with a grey background. The form contains several input fields: "Alias" (text), "Nombre" (text), "Fecha de Nacimiento" (date picker), "Mail de contacto" (text), and "Clase de Bono" (dropdown menu). There is also a checkbox for "Acepto condiciones" and two buttons at the bottom: "Enviar" and "List".

Integración de javascript

Javascript nos permitirá interactuar con nuestros controles que hemos definido. Todos los controles pueden tener un ID que los identifica. Este id es el que utilizaremos desde javascript para interactuar y manipularlo. Para poder insertar código nativo (html,css,javascript) lo podemos hacer usando las etiquetas HTML <oContenedor>/ ENDTEXT

tutor11.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'Get' INIT

    DEFINE FORM o

    INIT FORM o

        ROWGROUP o

            GET ID 'myid'          VALUE '123' GRID 4 LABEL 'Id.' ;
                PLACEHOLDER 'User Id.' ;
                BUTTON 'GetId' ACTION 'GetId()' OF o

        END o

        HTML o
            <script>

                function GetId() {

                    var cId = $('#myid').val()

                    MsgInfo( cId )

                }

            </script>
        ENDTEXT

    END FORM o

retu nil
```


Este ejemplo hemos de observar los siguientes puntos:

- El control tiene un ID para identificarlo
- La acción a ejecutar del button hace referencia a una función en Javascript
- Insertamos javascript con el comando HTML

Esta es la manera en que codificaremos la parte de javascript con Tweb

Ejecutar Formulario

Básicamente podremos ejecutar un formulario de 2 maneras

Via Form

Esta es la forma clásica de enviar datos al servidor. Para poder enviar los datos hemos de definir en el comando FORM el prg que ejecutaremos cuando enviemos el form con la clausula ACTION

DEFINE FORM o ACTION 'tutor12_srv.prg'

También habremos de definir el disparador de la acción que normalmente será un button al que especificaremos la clausula SUBMIT

BUTTON ID 'mybtn' LABEL 'Enviar' GRID 4 SUBMIT OF o

Cuando se ejecute el submit enviara via "post" (podemos cambiar el método con la clausula METHOD en el form) todos las variables que hemos identificado en cada control con la clausula ID

tutor12.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'Tutor 12' INIT

    DEFINE FORM o ACTION 'tutor12_srv.prg'

        INIT FORM o

            ROWGROUP o
```

```
GET ID 'myid' VALUE '123' GRID 4 LABEL 'Id.' OF o
GET ID 'myphone' VALUE '567' GRID 4 LABEL 'Phone' OF o

END o

ROWGROUP o
    BUTTON LABEL 'Enviar' GRID 4 SUBMIT OF o
END o

END FORM o

retu nil
```

Este código enviara el contenido de los 2 gets, cada get tendrá el nombre de la variable ID.

La 2 parte del código que es la que recibe los parámetros y responde podría ser algo similar a:

```
function main()

    ? 'Parameters<hr>'

    ? AP_PostPairs()

retu nil
```

Simplemente lo que hacemos es recoger los parámetros con la función de modharbour ap_postpairs() y ver lo que nos llega

Y esto es todo !

Via Ajax

La otra opción es la de manipular el dom, recuperar desde javascript el valor de las variables, enviarlas via Ajax, y esperar la respuesta del servidor para procesarla

tutor13.prg

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'Tutor 13' INIT
```

```
DEFINE FORM o
  INIT FORM o
    ROWGROUP o VALIGN 'bottom'
      GET ID 'myid' VALUE '123' GRID 4 LABEL 'Id.' ;
        PLACEHOLDER '123 for test...' ;
        BUTTON 'Search' ACTION 'GetId()' OF o
      GET ID 'myphone' VALUE '' GRID 4 READONLY OF o
    END o
  HTML o
    <script>
      function GetId() {
        var cId = $('#myid').val()
        $( '#myphone' ).val( '' )
        var oParam = new Object()
          oParam[ 'myid' ] = $('#myid').val()
        MsgServer( 'tutor13_srv.prg', oParam, PostGetId )
      }
      function PostGetId( dat ) {
        console.log( dat )
        if ( dat.success )
          $( '#myphone' ).val( dat.phone )
        else
          MsgError( dat.error, 'Response from Server' )
      }
    </script>
  ENDTEXT
END FORM o

retu nil
```

La 2 parte del código que es la que recibe los parámetros y responde podría ser algo similar a:

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

function main()

    local uValues      := GetMsgServer()
    local cPhone       := ''
    local lSuccess      := .f.

    AP_SetContentType( "application/json" )

    if uValues[ 'myid' ] == '123'
        lSuccess      := .t.
        cPhone        := '(34) 696948920'
    endif

    if lSuccess
        ?? hb_jsonEncode( { 'success' => lSuccess, 'phone' => cPhone } )
    else
        ?? hb_jsonEncode( { 'success' => lSuccess, 'error' => 'ID not found!' } )
    endif

    retu nil
endfunction
```

En esta parte de código podemos observar el uso de la función `GetMsgServer()` que devuelve un hash con todos los parámetros enviados desde el cliente con la función `MsgServer()`. Hay ejemplos de uso para ver como funciona

Controles

Llamaremos controles a los diferentes inputs, etiquetas y componentes del DOM usando sintaxis xBase para un uso mas familiar y amigable. Así pues tenemos los siguientes controles:

SAY, GET, BUTTON, CHECKBOX, FOLDER, FORM, MEMO, RADIO SELECT, SWITCH, BROWSE,...

Muchos de estas clases comparten propiedades que son comunes a todas y otras tienen propiedades propias, pero básicamente el conocimiento de una ayuda a entender las demás.

Cuando creamos un objeto, este llevará un ID para identificarlo. No es obligatorio pero necesario si queremos interactuar posteriormente por ejemplo con javascript, pues es el id que identifica el elemento en el dom. Un ejemplo sería este:

```
GET ID 'myid' LABEL 'Id.' VALUE "" GRID 6 OF oForm
```

Los parámetros pueden ser opcionales o obligatorios dependiendo de cada tipo de control pero siempre tendremos de especificar el control a que contenedor o ámbito pertenece, en este caso <oForm>, por lo que la cláusula OF <oContenedor> siempre será obligatoria.

Otra de las cláusulas comunes es la de GRID (o COL) que indicará cuantas de las 12 columnas ocupará el control

Clausulas Comunes

Cláusula	Descripción
ID	Identificador del control
GRID, COL	Numero de las 12 columnas que se van a tomar el control
OF	Referencia al contenedor (Obligatorio)
ONCHANGE	Función que ejecutaremos cuando haya un cambio de valor en el control. ONCHANGE "MyControl()"
ALIGN	Algunos controles pueden llevar esta cláusula de alineamiento horizontal: Left, center, right
FONT	Objeto Font que usamos para definir la fuente del control
CLASS	Regla CSS que se añada al control en caso de querer usarla
VALUE	Valor

GET

Cláusula	Descripción
LABEL	Etiqueta del Get
READONLY	Input de lectura
TYPE	Valores type de un input. https://www.w3schools.com/tags/att_input_type.asp
BUTTON	Button asociado al get. Especificaremos un texto. BUTTON "Search"
ACTION	Función Javascript (JS) que se ejecutará al pulsar el button
REQUIRED	Entrada obligatoria
AUTOCOMPLETE	Busca en una lista que podra ser un array o un prg al que le enviaremos los datos entrados para que realice una búsqueda y nos devuelva un array. Ver ejemplos de tipo get.
SELECT	Va en complemento a AUTOCOMPLETE. Si se especifica con el nombre de una función esta se ejecutara cuando se seleccione un elemento de la lista de autocomplete
PLACEHOLDER	Literal que aparecera cuando el campo este vacio
MEMO	El tipo de control será un textarea para editar textos

FONT

Cláusula	Descripción
NAME	Nombre de la clase. Equivalente a ID
COLOR	Color de la fuente
BACKGROUND	Color del fondo
SIZE	Tamaño
BOLD	Negrita
ITALIC	Itálica

RADIO, SELECT

Cláusula	Descripción
ITEMS, PROMPTS	Array de valores que se muestran
VALUES, KEYS	Array de valores con la clave de los ITEMS. Es el valor del elemento seleccionado que devuelve el control cuando lo solicitamos
INLINE	Radios en linea

FOLDER

Cláusula	Descripción
PROMPTS, ITEMS	Array de literales que se muestran en cada Tab
TABS	Array con el ID de los contenedores de cada Tab
OPTION	Tab inicial
ADJUST	Ajustar Folder al contenedor

El Folder viene acompañado de una segunda clausula que es cuando definimos el contenedor de cada TAB. Dentro de cada Contenedor podemos definir como en todos los demás todos los controles que necesites. Acabaremos la definición del TAB con ENDTAB <oFld>

```
DEFINE TAB 'menu1' OF oFld  
  
    GET oGet ID 'myid' VALUE '' LABEL 'Id. User' BUTTON OF oFld  
  
ENDTAB oFld
```

BROWSE / GRID

Este capítulo lo dedicaremos a este control quizás porque es el mas complejo y ofrece una gran variedad de comandos y funciones. En internet encontraremos numerosas librerías para manejar un browse y algunos de muy buenos: TDatatables, Slickgrid,...pero para Tweb usaremos Bootstrap-table (<https://bootstrap-table.com/>) porque se adapta perfectamente a Bootstrap y es muy sencillo de usar.

Lo primero que hemos de saber es que si vamos a usar el grid vamos a tener que cargar las librerías. Para esto hemos de indicar en la definición de web la clausula TABLES

```
DEFINE WEB oWeb TITLE 'Test Browse' TABLES INIT
```

Esto es suficiente para que Tweb sepa que ha de cargar todos los ficheros necesarios para usar el browse

Primero de todo habremos de definir el browse como un control mas usando nuestra sintaxis típica

```
DEFINE BROWSE oBrw ID 'ringo' HEIGHT 400 OF o
```

Observad que siempre el control va asociado a un contenedor

Definimos las columnas que va a formar el browse con el siguiente comando

```
ADD oCol TO oBrw ID 'id'    HEADER 'Id.'  
ADD oCol TO oBrw ID 'name' HEADER 'Name'
```

Finalmente inicializaremos el browse. Es posible inicializarlo y asignarles directamente ya los datos

```
INIT BROWSE oBrw DATA aRows
```

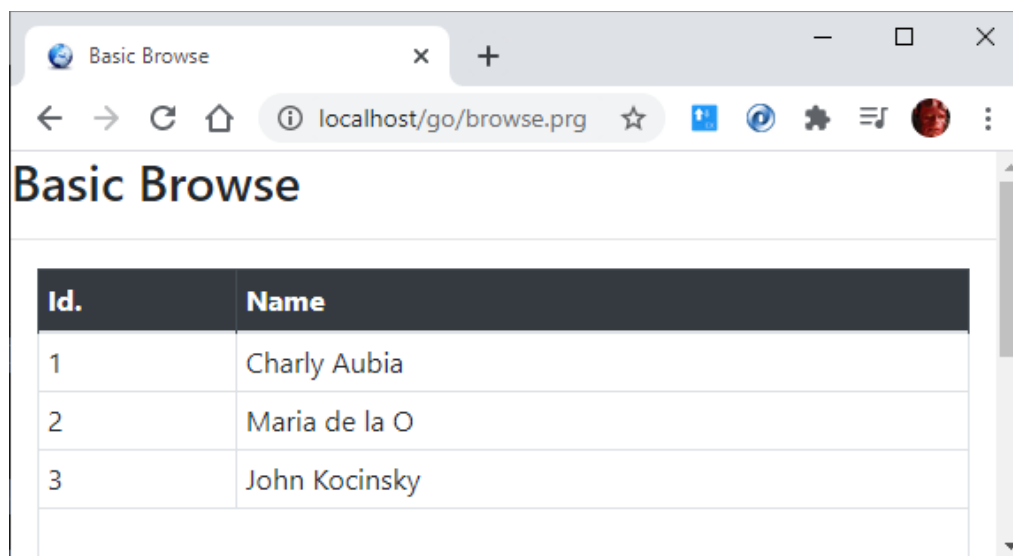
Los datos será una variable de tipos Array con valores hash que indicaran columna/valor

```
Aadd( aRows, { 'id' => 1, 'name' => 'Charly Aubia' } )  
Aadd( aRows, { 'id' => 2, 'name' => 'Maria de la O' } )  
Aadd( aRows, { 'id' => 3, 'name' => 'John Kocinsky' } )
```

Y esto es la base para crear un browse. Un ejemplo básico sería el siguiente

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}  
  
#include {% TWebInclude() %}  
  
function main()  
  
    local o, oCol, oBrw  
    local aRows := {}  
  
    Aadd( aRows, { 'id' => 1, 'name' => 'Charly Aubia' } )  
    Aadd( aRows, { 'id' => 2, 'name' => 'Maria de la O' } )  
    Aadd( aRows, { 'id' => 3, 'name' => 'John Kocinsky' } )  
  
    DEFINE WEB oWeb TITLE 'Basic Browse' TABLES INIT  
  
    DEFINE FORM o  
  
        HTML o INLINE '<h3>Basic Browse</h3><hr>'  
  
    INIT FORM o  
  
        DEFINE BROWSE oBrw ID 'ringo' HEIGHT 400 OF o  
  
            ADD oCol TO oBrw ID 'id'    HEADER 'Id.'  
            ADD oCol TO oBrw ID 'name'  HEADER 'Name'  
  
            INIT BROWSE oBrw DATA aRows  
  
        END FORM o  
  
    retu nil
```

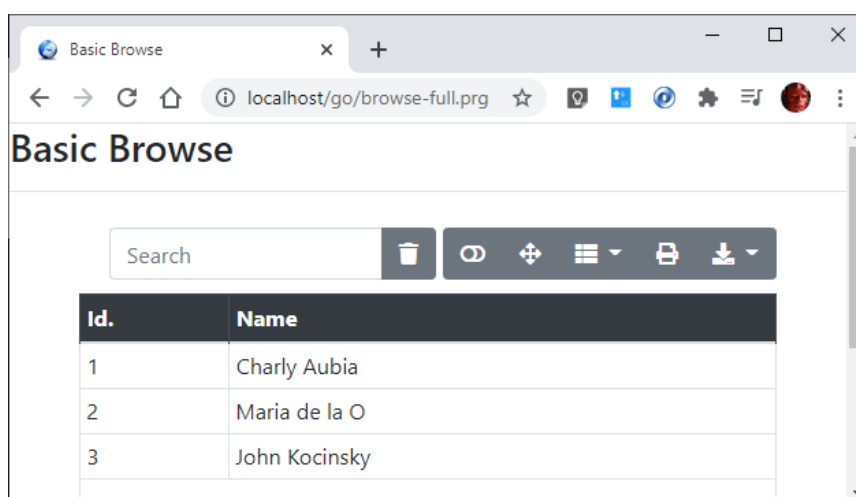

Y el resultado seria este



Un browse lo podemos definir e ir añadiendo funcionalidades: controle de eventos (como dblclick), formateador de columnas, ... Toda esta parte de código la crearemos en Javascript. Si lo necesitamos podemos inicializar el browse dentro de javascript. Es normal definir en javascript las funciones que p.e. se encargaran de crear peticiones al servidor y funciones que "escuchen" la respuesta y añada los registros al browse. Hay muchos ejemplos en la que se puede ver como de manera sencilla podemos gestionar este escenario.

La definición del Browse tienes muchas clausulas opcionales que nos añadirá si necesitamos de una manera muy fácil numerosas funcionalidades: vista de registro, búsqueda, toda la pantalla, impresión, exportación,... simplemente poniéndolo en la misma línea de definición

DEFINE BROWSE oBrw ID 'ringo' HEIGHT 400 EXPORT SEARCH PRINT TOOLS OF o



Los comandos que pueden usarse para definir el browse son

Cláusula	Descripción
HEIGHT	Altura del browse
SELECT	Habilita columna con select
MULTISELECT	Igual que la anterior pero multi seleccion
CLICKSELECT	Cualquier parte de la línea sirve para seleccionar
PRINT	Button de Impresión
EXPORT	Button de exportación
SEARCH	Activa modo búsqueda
TOOLS	Ocultar columnas
ONCHANGE	Función javascript que se ejecutara en cada change
DBLCLICK	Función javascript que se ejecutara con el dblclick
EDIT UNIQUEID <cId> TITLE <cTitle> POSTEDIT <cPostEdit>	EDIT Activa modo edición de registro UNIQUEID <id> Definimos el campo único que marcará un registro TITLE <cTitle> Si lo especificamos aparecerá en el dialogo de edición POSTEDIT <cPostEdit> Sera la función que se ejecutara en el caso que actualizemos la edición. Se pasará de parámetros el registro actualizado
ROWSTYLE <cRowStyle>	Función que evaluara todo el registro para poderle aplicar una regla css
TOOLBAR <cId>	Podremos poner un div a la altura de la toolbar

La clase TWebBrowse tiene otras propiedades como

Cláusula	Descripción
IDark	Efecto Dark
IStripped	Efecto Stripped
cLocale	Idioma usado en el browse. Defecto EN. Podéis ver toda la lista de valores en: https://examples.bootstrap-table.com/

Los comandos que define la columna pueden ser los siguientes

Cláusula	Descripción
HEADER	Cabecera de la columna
WIDTH	Ancho de la columna
ALIGN	'left', 'center', 'right'
FORMATTER	Function javascript que se encargará de formatear la columna
SORT	Activar ordenación por columna
CLASS	Podemos devolver el nombre de una class que se aplicará a la col: return { classes: 'myid' } También podemos devolver el css directamente

	<pre>return { css: { 'background-color': 'lightgray', 'font-weight': 'bold' } }</pre>
EDIT	Indicamos que esa columna es editable
TYPE	Tipo de datos para poderlo editar correctamente. Valores: "L" → Lógico "D" → Date (Es importante que el campo este formateado YYYY-MM-DD) "M" → Memo "N" → Numérico "COMBOBOX" WITH <hParam> → Tipo lista. hParam será un hast key/value

Finalmente comentar que el Browse se puede manipular desde JavaScript cuando queramos realizar operaciones con el. Para crear una instancia del browse lo haremos de la siguiente manera

```
var oBrw = new TWebBrowse( <cld> )
```

Una vez creado el objeto podemos usar los siguientes métodos

Método	Descripción
SetData(rows)	Inicializar con un array de datos el browse
Clean()	Vaciar los datos. Mismo efecto que SetData()
Select()	Devuelve un array de los rows seleccionados en el caso de que usemos la clausula SELECT
SelectIndex()	Devuelve un array de los index seleccionados en el caso de que usemos la cláusula SELECT
GetData()	Devuelve el array de datos
GetDataChanges()	Devuelve un array con los registros que se han modificado. Tiene el formato: {acción, id, row} Acción puede tener los siguiente valores: 'A' → Nuevo registro 'D' → Registro eliminado 'U' → Registro actualizado "id" será el valor del campo UNIQUEID que nos marcar la posición en la tabla/base de datos
ResetChanges(<aChanged>)	Vaciar la tabla de cambios. Si pasamos <aChanged> que será un array devuelto por el server vaciara solo los registros de esta tablas. En caso contrario la vacia toda. Es la manera de sincronizar el browse con las actualizaciones que hacemos en el servidor
GetItemEmpty()	Devuelve un item vacío en base a la estructura que hemos definido
Edit()	Activa dialogo edición

AddRow()	Suma registro a la tabla. Importante tener clausula UNIQUEID definida
DeleteRow()	Eliminar posición de la tabla
GetRow()	Devuelve contenido del row/s marcados
AddRow(oltem, nIndex)	Añadir oltem a la tabla
UpdateRow(index, row)	Actualiza <row> en la posición de <index>
UpdateRowById(<cRowId>, row)	Actualiza <row> en la posición de <cRowId>
DeleteRow(alds, cMode)	Elimina row marcado por el registro alds o array de registros. cMode por defecto es "id" y hará referencia al UNIQUEID. Si especificamos "index" hará referencia al index de lors rows
Loading(<LOnOff>)	Muestra un mensaje de Loading

El control browse es uno de los mas complejos y aun que esta clase sea sencilla de usar la mejor manera de aprender el uso de este control es probando los numerosos ejemplos que hay sobre browse y experimentar con ellos

Otros controles

Hay otros controles que no están en la lista porque usan las clausulas comunes: SAY, CAPTION, SEPARATOR, SWITCH

Siempre podremos consultar el fichero de configuración del preprocesador tweb.ch para ver las definiciones de todos los controles.

Complementos de interés

Get Autocomplete

Una de las opciones interesantes para buscar información desde el servidor es el uso de la clausula AUTOCOMPLETE del control GET. Una vez la especifiquemos tenemos 2 opciones:

- Especificar un array de palabras → En función de lo que escribamos se abrirá una lista con las palabras que coincidan con lo escrito
- Especificar una url al server. Las palabras que estamos escribiendo las mandamos vía Ajax al server. Una vez el server las recibe buscamos o procesamos como queramos esta información. El resultado lo pondremos en una tablas de hash con tantas keys como queramos, pero ha de existir una key en le hash que se llame "value" que es la que se mostrar enn la lista.

c) La palabra que se ha escrito en el Get, se envía vía post con el nombre "query" de la variable.

Este seria un ejemplo de como buscar en una tabla un cliente a partir de las entradas en el GET

```
#define PATH_DATA          HB_GetEnv( "PRGPATH" ) + '/data/'

function main()

    local aRows := {}
    local cSearch := HB_HGetDef( AP_PostPairs(), 'query', '' )

    if empty( cSearch )
        AP_SetContentType( "application/json" )
        ?? hb_jsonEncode( {=>} )
    endif

    USE ( PATH_DATA + 'test.dbf' ) SHARED NEW VIA 'DBFCDX'
    SET INDEX TO ( PATH_DATA + 'test.cdx' )

    cAlias := Alias()

    (cAlias)->( OrdSetFocus( 'FIRST' ) )

    (cAlias)->( DbSeek( cSearch, .T. ) )

    WHILE (cAlias)->first = cSearch .and. (cAlias)->( !Eof() )

        Aadd( aRows, { 'id' => ltrim(str((cAlias)->( Recno() ))), ;
                        'value' => (cAlias)->first + ' ' + (cAlias)->last, ;
                        'street' => Alltrim((cAlias)->street), ;
                        'city' => Alltrim((cAlias)->city ), ;
                        'zip' => Alltrim((cAlias)->zip ) ;
                      } )

        (cAlias)->( DbSkip() )

    END

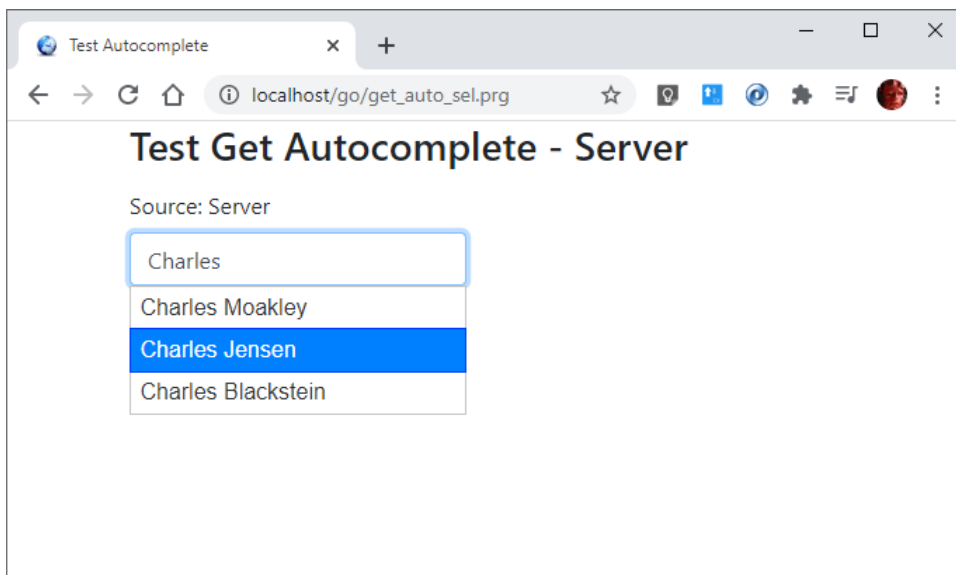
    AP_SetContentType( "application/json" )
    ?? hb_jsonEncode( aRows )

retu nil
```

La lógica es muy sencilla:

- Recuperamos el valor entrado en el get
- Abrimos la tabla
- Buscamos en la tabla coincidencias
- Cargamos resultados en array

- Devolvemos formato json la tabla.



Para finalizar podemos combinar la clausula AUTOCOMPLETE con SELECT <cFunc>. En el momento de seleccionar un elemento de la lista, se ejecutará la función que hemos especificado pasando todo el elemento de la tabla que hemos seleccionado.

Upload Files

Una de las necesidades mas comunes en aplicaciones web es poder subir ficheros al cloud. Con Tweb fácilmente podemos realizar esta acción muy fácilmente. Añadiremos la clausula FILES al comando BUTTON y automáticamente cada vez que pulsemos el button nos aparecerá una ventana para la selección de ficheros.

BUTTON ID 'myupload' LABEL ' FILES ACTION 'UploadFile()' OF o

Es necesario especificar un ID porque inicializaremos el control desde Javascript, el parámetro FILES y la función que inicializará el sistema, en este caso UploadFile()

Después hemos de inicializar desde Javascript.

HTML o

```
<script>

function UploadFile() {

    var o = new TWebUpload( 'myupload', 'srv_upload.prg', Post_UploadFile )
```

```

    o.Init()
}

function Post_UploadFile( dat ) {

    console.log( 'Post_UploadFile', dat )

    MsgInfo( 'File uploaded' )
}

</script>

ENDTEXT

```

La función de inicialización usará la clase TWebUpload() con los siguientes parámetros:

TWebUpload(cId, cUrl, bCallback, oParameters)

Parámetros	Descripción
cId	ID del button
cUrl	Url del servidor para ejecutar el programa que control la subida
bCallback	Función que se ejecutara cuando recibamos una respuesta del servidor
oParameters	Opcional. Objeto de parámetros que deseamos enviar también al servidor además del fichero

El objeto que creamos con TWebUpload le podemos asignar una serie de funciones a diferentes eventos que nos permite controlar el estado del proceso.

Event	Descripción
onprogress	Evento que nos indica el estado de progreso de la subida
onloadstart	Evento que se ejecuta al inicio
onloadend	Evento que se ejecuta al final
onreading	Evento que se ejecuta cuando leemos el fichero

Podéis ver 2 ejemplos, uno simple y el otro usando todas las posibilidades:

button-upload.prg y button-upload-ext.prg.

[Templates → Html File](#)

Podemos usar el uso de templates con Tweb de una manera sencilla. Cuando usamos de modo reiterada código igual, podemos poner esta parte de código en un fichero que nosotros podemos llamar desde nuestro programa, incluso pasándole parámetros. Vendría a ser similar al concepto de Vistas (Views) que se usa en Mercury o cuando usamos patrones de diseño como el MVC.

Ejemplo de un template → `template/card.tpl`

```
<div class="card" style="width:400px">
  
  <div class="card-body">
    <h4 class="card-title"><$ parameter( 'id' ) $></h4>
    <p class="card-text"><$ parameter( 'name' ) $></p>
    <a href="#" class="btn btn-primary">See Profile</a>
  </div>
</div>
```

Este es un código cualquiera, en este caso un ejemplo que he hecho un copy/paste de https://www.w3schools.com/bootstrap4/bootstrap_cards.asp, y que podría ser cualquiera. En este pequeño snippet le he insertado los tags de 3 variables que pasaré cuando llame al template

`<$ parameter('name') $>`

Ya solo falta ver como desde nuestro código llamamos al template

```
//      {% LoadHrb( 'lib/tweb/tweb.hrb' ) %}

#include {% TWebInclude() %}

function main()

    LOCAL o, oWeb

    DEFINE WEB oWeb TITLE 'Test template' INIT

    DEFINE FORM o

    HTML o INLINE '<h4>Test Profile...</h4><hr>'

    INIT FORM o

    ROW o HALIGN 'center'

        HTML o FILE 'templates/card.tpl' ;
            PARAMS { 'id' => 'ACF-7639',;
                    'name' => 'John Kocinsky',;
                    'photo' => 'images/user.jpg' }

    END o
```



```
END FORM o  
retu nil
```

Podemos pasar los parámetros como variables separados por coma, o con un hash. Si separamos con coma, en el template los recuperamos en el mismo orden: `parameter(1)`, `parameter(2)`,...

```
HTML o FILE 'templates/card.tpl' PARAMS 'ACF-7639', 'John Kocinsky', ...
```

Mientras que si especificamos un hash, lo recuperamos por la key: `parameter('name')`, `parameter('id')`,...

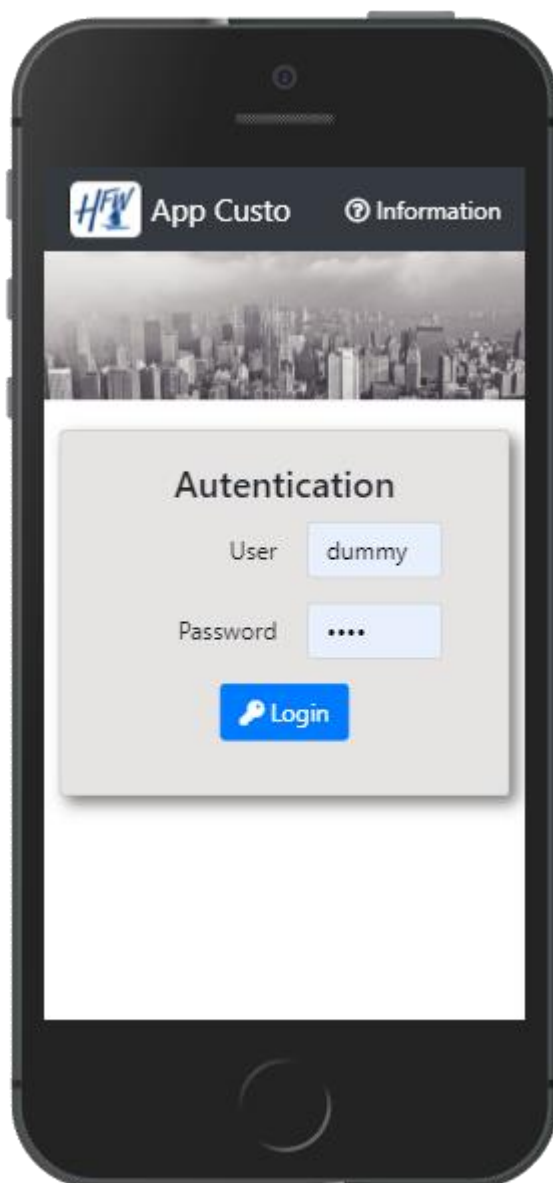
Al final podemos pensar que es como llamar a una función, es el mismo concepto



Esto es muy útil para el uso de partes de código redundante. Manteniendo solo la parte del template ya afectará en cualquier parte de nuestra aplicación donde usemos este template.

Utilizando esta técnica al principio, es muy fácil coger porciones de código de ejemplo de la web y crear pantallas muy vistosas.

Siguiendo este mismo ejemplo del "card" si observamos tienes 2 divs que se crean con otros 2 div que los cierra. Podríamos crear una plantilla que abriéramos una card (como si fuera una box), luego insertar nuestros controles y finalmente cerrar la card. Imaginaros una pantalla de login para el smartphone



El código esta en los ejemplos y es muy fácil de entender → pone-login.prg

Utilidades Javascript

Estas son alguna de las funciones que pueden servir de complemento

Function	Descripcion
SetDataJS(uValue)	Convierte un valor PRG a JavaScript en el formato que tiene original. var cValue = <\$ SetDataJS(cString) \$> Ver ejemplo prgtojs.prg
MsgInfo(cMsg, cTitle, clcon, fCallback)	Mostrar mensaje <cMsg> en pantalla. Opcionalmente: cTitle → Titulo ventana clcon → Icono que se muestra con el titulo en formato fontawesome: https://fontawesome.com/icons?d=gallery fCallback → función javascript que se ejecutara cuando salgamos del MsgInfo()
MsgError(cMsg, cTitle, clcon, fCallback)	Misma funcionalidad que MsgInfo()
MsgYesNo(cMsg, cTitle, clcon, fCallback)	Misma funcionalidad que MsgInfo(), Envia a la function true/false
MsgNotify(cMsg, cType)	Mensaje de notificación. Valores cType: success, warning, danger, error

Charset

Es muy importante en la web codificar al mapa de caracteres que vamos a utilizar. Básicamente usaremos 2: Latin-1 y UTF-8, aunque ultimamente la tendencia es usar siempre Utf-8

En resumen podríamos definir que Latin-1 cubre la mayoría de lenguajes de Europa Occidental como el albanés, catalán, danés, neerlandés, inglés, feroés, finés, francés, alemán, gallego, gaélico, islandés, italiano, noruego, portugués, español y sueco.

Si no vamos a usar ningún carácter/símbolo que se salga de este "mapa de caracteres" podríamos trabajar perfectamente en Latin-1. Pero ¿qué pasa si queremos usar otros caracteres?

Este documento de texto está codificado en utf-8 por lo que me permite poner los siguientes caracteres:

Special characters ÄäÖöÜ Barça caña àáèéìóú ü î

中国版 (China)

香港版 (Hong Kong)

日本 (Japan)

한국 (Korea)

台灣版 (Taiwan)

ישראל (Israel)

Ε λ λ α δ α (Greece)

العالم العربي (Arabic)

Р о с с и я (Russia)

हिन्दी (India)

தமிழ் (India)

മലയാളം (India)

తెలుగు (India)

Si esta fuera mi página web estaría codificada en utf-8 por lo que la página hemos de especificar que usaremos utf-8. Si la página no se especificara utf-8 estos caracteres especiales los intentaría traducir y se verían mal. Aquí ya podemos deducir pues que si codificamos en utf-8 la página debe estar en charset="utf-8"

```
DEFINE WEB oWeb TITLE 'Test CharSet - UTF8' CHARSET 'utf-8' INIT
```

Podéis probar el ejemplo charset-a.prg codificado en utf8 y con charset="utf8"

En este otro ejemplo se puede ver como la página la hemos codificado en 'utf-8' pero en el código hemos especificado que use Latin-1.

```
DEFINE WEB oWeb TITLE 'Test CharSet - UTF8' INIT
```

Por defecto el comando DEFINE WEB usa el charset 'latin-1' por lo que no ha de especificar la clausula charset. La página en nuestro editor se ve bien en nuestro editor pero cuando la ejecutemos en la web, este intentara codificar en latín-8 símbolos que hemos codificado en utf-8 en nuestro editor y se verá mal. Podéis ver el resultado en charset-a-ko.prg

Ya podemos apreciar la importancia de saber codificar bien nuestras paginas para no sufrir estos efectos adversos que tantos dolores de cabeza nos da a los programadores.

Si probáramos de hacer copy/paste de estos símbolos de mas arriba a una pagina codificada en ANSI nos aparecería esto:

Special characters ÄäÖöÜ Barça caña ààèèìòóú ü î

??? (China)
??? (Hong Kong)
?? (Japan)
?? (Korea)
??? (Taiwan)
???? (Israel)
????da (Greece)
?????? (Arabic)
?????? (Russia)
?????? (India)
?????? (India)
?????? (India)
?????? (India)

Como podemos observar, los primeros los acepta correctamente pero los otros no, porque no los sabe codificar. Imaginemos ahora que en nuestra pagina codificada en ANSI nos quedamos con los primeros

Special characters ÄäÖöÜ Barça caña ààèèìòóú ü î

Si nosotros creamos la web con este fichero en ANSI y la pagina no especificáramos nada, la pagina nos la mostraría correctamente. Pero en cambio si la página especificáramos que use un charset='utf-8' intentaría traducir los caracteres codificados en ANSI a UTF-8 y se verían mal !!!

Podéis probar el ejemplo charset-b.prg codificado en ANSI y definimos charset="Latin-1"

También podéis ver un típico error de codificar nuestro código en ANSI y utilizar un charset='utf-8' en el ejemplo charset-b-ko.prg

Resumen:

- Si codificamos en ANSI -> página en Latin-1
- Si codificamos en UTF8 -> página en utf-8

Charset – Dbf

Últimamente la tendencia en el mundo web es codificar la página con utf-8 y así podemos usar cualquier carácter especial sin problemas alguno y así es. Pero entran en acción nuestras dbf que nos rompen nuestros esquemas...

Una vez vista la primera parte podemos pensar de varias maneras acorde a nuestras necesidades. Unos pueden pensar que todo en utf8 y así no habrán problemas, otros prefiriendo el Latin-1 porque para usar su idioma ya tiene suficiente...

En la mayoría de los casos nuestras dbf están en ISO-8859-1 entonces que ocurre si tenemos una tabla dbf en la que hemos salvado nuestros caracteres típicos especiales como acentos, ç, ñ, ... y nuestra página esta usando utf-8 ? → charset-c-ko.prg

Es por eso que si la mayoría de las veces usamos el charset='Latin-1' tendremos suficiente para poder manipular nuestras dbfs → charset-c.prg

Pero puede ocurrir que tengamos que usar una tabla en la que hayan salvado en utf8, y entonces nuestra página en latin-1 ya no se adaptaría bien → charset-d-ko.prg

En este caso, si nosotros tenemos un charset='utf-8' y usamos esta tabla ya podríamos ver aquellos caracteres especiales que no podríamos acceder con el charset='Latin-1' → charset-d.prg

A partir de aquí se nos abre los primeros dilemas de como debemos y que estrategia seguir para poder codificar y salirse victoriosos con nuestros programas. Hay quien dice que quiere utilizar utf8 porque es la tendencia (no su necesidad) y que al leer de los dbf convertir el contenido a utf-8 para que se muestre bien en la pantalla (vaya perdida de rendimiento, no?). Cada uno es libre de usar lo que mejor crea, en este capítulo se ha explicado los pros y contras y todo lo que os pueda ocurrir.

Finalmente, podemos manipular y convertir la información que leemos/escribimos en nuestras tablas con funciones que convirtieran (escapar) estos caracteres, pero ya lo veremos en otro capítulo...

Conclusiones

Tweb es una librería diseñada para ayudar a los programadores que dan el salto a la web. Mientras la usamos nos iremos familiarizando con diferentes conceptos de la web, css, funcionamiento de Bootstrap... Llegará el momento en que ya no la necesitemos o la queramos seguir usando juntamente con código nativo.

La velocidad de diseño de paginas y la productividad que nos ofrece es muy alta, por lo que el gran problema que teníamos al inicio de todo el proceso ya lo tendremos controlado.

En esta primera versión se ha sentado unas base para poder trabajar bien con la librería y esta diseñada para escalarla fácilmente con mas controles y funcionalidades

Aprovechemos la gran funcionalidad que nos proporciona nuestro preprocesador y usémosla para el diseño de nuestras pantallas...