COBHTTPD

Servidor de Aplicaciones para COBOL.

Manual de Desarrollo

(Ver. 4.80)

I N D I C E

Descripción.

En este manual se explicara el funcionamiento de las librerías COBLIB.
(Anteriormente llamadas CGILIB)

Estas librerías coblib son una serie de Variables y Parrafos para COBOL, para que de esta manera sea mas fácil leer las variables que provienen de la Web y generar la salida HTML para enviar al cliente remoto.

Ejecutando los programas COBOL

Existen dos maneras de ejecutar un programa COBOL. El modo a utilizar dependerá del tipo de contenido a regresar.

Generando Contenido HTML:

Si deseamos regresar contenido HTML usaremos:
http://www.tudominio.com/cobapp?proid=PRO&prog=PROG&F1=value

Generando Reportes PDF:

Si deseamos regresar un report PDF a el navegador, utilizaremos el modo:
http://www.tudominio.com/cobpdf?proid=PRO&prog=PROG&F1=value

Acceso directo:

Desde el campo dirección, en el web browser, puedes correr un programa de la siguiente manera:

http://www.tudominio.com/cobapp?proid=PRO&prog=PROG&F1=value

Con este dato, puedes correr un programa llamado PROG en el proyecto PRO, y recibirá el campo F1 con el valor indicado en value.

Recuerda que, si definiste un puerto diferente al 80, entonces necesitas indicar este puerto en tu Server, ejemplo, si defines el puerto 85:

http://www.tudominio.com:85/cobapp?proid=XXXXXXX

Llamando desde archivos HTML

Para una llamada desde formas html, solo coloca la dirección en el ACTION de una marca FORM:

```
<form method="POST" action="/cobapp?proid=PRO&prog=PROG”>
  … definición de campos de la forma
</form>
```

Para gente que conoce poco de html, si ves en el action, podrás ver que no escribí el url completo. Esto no es necesario, por que el browser se encarga de expandir este url con el nombre del dominio de donde descargo este html, y con el puerto del webserver. A esto se le conoce como dirección relativa.

Pero, si tienes el html de la forma en un web Server diferente, y solo buscas correr tus programas COBOL con el servidor cobhttpd, entonces tu POST podría verse así:

<form method="POST"
action="http://www.tudominio.com/cobapp?proid=PRO&prog=PROG”>
… definición de campos de la forma
</form>

A esta dirección se le conoce como dirección absoluta.

Si quieres pasar un dato, pero este dato no tiene campo INPUT dentro de la forma html, puedes pasar este dato como parte del url, ejemplo:

<form method="POST" action="http://www.tudominio.com/cobapp?
proid=PRO&prog=PROG&func=add”>

En este ejemplo, el campo **func**.

Este campo es pasado al programa PROG, y este programa necesita tener definido este campo (func) en su lista de campos también.

Cuando nuestro programa COBOL se ejecute, func también le llegara, igual que los campos que están definidos en la forma HTML con las marcas INPUT.

Diferencia entre contenido HTML y PDF

Para contenido html usar **/cobapp?proid=proyecto&prog=prog**
Para contenido pdf usar **/cobpdf?proid=proyecto&prog=prog**

En ambos casos el archivo a generar debe de llamarse tal y como nos indique la variable: HTTP-OUTPUT (esta variable se explica mas adelante en el registro HTTP-CONTROL).

Recibiendo datos de entrada en nuestro programa COBOL

Hasta este momento, nuestro programa COBOL recibe dos registros de datos:

```
01  HTTP-CONTROL.
    02 HTTP-PROJECT     PIC X(20).
    02 HTTP-PROGRAM     PIC X(20).
    02 HTTP-OUTPUT      PIC X(150).
    02 HTTP-ERROR       PIC 9(04).
    02 HTTP-ERROR-MSG   PIC X(50) OCCURS 5 TIMES.

01  HTTP-DATA-INP.
    02 HTTP-DATA-INPX     PIC X(64000).
```

Estos dos registros se encuentran ya definidos en el archivo **LS-HTTP.FDT** que se encuentra en el directorio COBLIB dentro de COBHTTPD, también va en cada ejemplo ya que se necesita para compilar los programas.

HTTP-CONTROL

El primer registro: HTTP-CONTROL contiene algunos datos importantes para nuestro programa, y que son pasados de COBHTTPD hacia nuestro programa COBOL que será ejecutado:

HTTP-PROJECT – es el código del proyecto al que pertenece nuestro programa que se esta ejecutando. Nos puede servir dentro de nuestro programa para hacer enlaces hacia urls y colocar el mismo código de proyecto que en el archivo html externo.

HTTP-PROGRAM – es el nombre del programa que se esta ejecutando, y nos puede servir para hacer librerías externas y hacer referencia hacia el programa que se esta ejecutando, y no tener que poner el nombre del programa fijo, si es que nuestra librería sirve para diferentes programas.

HTTP-OUTPUT – Es el nombre del archivo que debemos de generar con nuestro contenido html que deseamos.

HTTP-ERROR – Es un código de error que podemos regresar a COBHTTPD, cualquier numero diferente de 0 es un error (no usado por el momento).

HTTP-ERROR-MSG – Es un arreglo de 5 renglones donde podremos colocar un mensaje textual para regresar al cliente en caso de que el programa necesite informar algún error, al ser este HTML es necesario regresar el error como parámetro. Trabaja en conjunto con HTTP-ERROR. (no usado por el momento).

HTTP-DATA-INP

Este segundo registro, es un solo campo de 64,000 bytes de tamaño. En este campo nos llegaran los diferentes valores que nuestro programa COBOL necesita para funcionar.

Al ser de un máximo de 64,000 podemos incluso recibir grandes cantidades de datos para realizar operaciones de Actualización de archivos.

Ya en nuestro programa hacemos la división de campos declarando un registro en la WORKING-STORAGE, y solo movemos este registro a nuestro registro interno para tener la división real. Ejemplo:

```
WORKING-STORAGE SECTION.

01  REG-INTERNO.
    02  MFUNC            PIC X(15).
    02  MKEY-INI         PIC 9(05).
    02  MKEY-FIN         PIC 9(05).
    02  BKEY             PIC 9(05).

PROCEDURE DIVISION.
INICIO.
   MOVE HTTP-DATA-INP  TO REG-INTERNO.
```

De esta manera usamos los campos de REG-INTERNO en nuestro programa.

Para indicarle que variables vamos a recibir de la web, también hay que indicarlo en la configuración del Proyecto, esto viene explicado en el manual del usuario de cobhttpd, donde se explica todas las funciones del programa cfgcobhttpd.

Como generar la salida a entregar al cliente sin usar COBLIB

Ya sea generando contenido HTML o generando un reporte para ser entregado como PDF debemos de grabar el contenido en un archivo texto.

COBHTTPD nos indicara como se debe de llamar el archivo texto que debemos generar. Este nombre de archivo nos llegara al programa en la variable HTTP-OUTPUT. Esta variable tiene el nombre del archivo. Si no generamos el contenido con este nombre el resultado no será entregado al cliente que solicito la consulta.

Ahora, desde COBOL podemos utilizar la siguiente declaración para crear este archivo:

```
… …
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT IMPRESO
           ASSIGN TO INPUT-OUTPUT WF-REPORT
           ORGANIZATION IS LINE SEQUENTIAL
           FILE STATUS IS STAT-IMP.
… …

DATA DIVISION.
FILE SECTION.
FD  IMPRESO
    LABEL RECORDS ARE OMITTED.
01  LINEA          PIC X(500).

WORKING-STORAGE SECTION.

77 WF-REPORT        PIC X(150).
77 STAT-IMP         PIC X.

… …
LINKAGE SECTION.

COPY "COBLIB\LS-HTTP.FDT".

PROCEDURE DIVISION USING HTTP-CONTROL HTTP-DATA-INP.
000-INICIO.
    MOVE HTTP-OUTPUT TO WF-REPORT.
    OPEN OUTPUT IMPRESO.

    WRITE LINEA FROM "<html><head>".
    WRITE LINEA FROM "<title>ejemplo html</title>".
    WRITE LINEA FROM "</head>".
    WRITE LINEA FROM "".
```

```
           WRITE LINEA FROM "<body>".
           WRITE LINEA FROM "<center>".
           WRITE LINEA FROM "<h1>Primer prueba html</h1>".
           WRITE LINEA FROM "</center>".
           WRITE LINEA FROM "</body>".
           WRITE LINEA FROM "</html>".

           CLOSE IMPRESO.
           STOP RUN.
       END PROGRAM.
```

En este pequeño ejemplo hacemos solamente un desplegado de un letrero que debe de decir:

Primer prueba html

Todas las marcas utilizadas aquí son html, asi que no debe de haber ningun problema con esa parte.

Lo interesante primero es:

- Utilizamos el archivo LS-HTTP.FDT para obtener todas las variables que nos entrega COBHTTPD
- Utilizamos el nombre del archivo que nos llega en HTTP-OUTPUT
- Creamos el archivo con el contenido deseado

Para crear contenido PDF es idéntico, solo que las marcas que usaremos para crear un reporte PDF deben ser las mismas marcas que para crear un reporte para COBVIEW. Hay que recordar que los reportes que se visualizan en Windows con COBVIEW pueden ser también creados para internet y visualizados en formato PDF que es la razón por la que se hace esta conversión.

Descripción de los archivos que componen COBLIB

COBLIB son archivos que podemos incluir en nuestros programas COBOL, y definen tanto variables como nombres de párrafo, y se encuentran comúnmente en el directorio C:\COBHTTPD\COBLIB (a menos que se hubiera instalado en un directorio distinto.

Los archivos son los siguientes:

HTML-OUT.SEL	Contiene el SELECT del archivo HTML-OUT, con el que podemos enviar el contenido de respuesta al cliente.
HTML-OUT.FD	Contiene la declaración del FD, del archivo HTML-OUT, con el que enviaremos la respuesta al cliente.
LS-HTTP.FDT	Son las variables de comunicación que envía COBHTTPD hacia nuestro programa COBOL, este archivo debe incluirse dentro de la LINKAGE SECTION.
TEMPLATE.WSS	Archivo que contiene toda la declaración de variables que se necesitan para enviar los diferentes comandos de COBOL a COBHTTPD.
TEMPLATE.FDT	Este contiene todos los párrafos para armar los comandos que nustro programa COBOL enviara al COBHTTPD.

Básicamente la idea de esta librería es poder facilitar el envió de información hacia el template html que se esté utilizando.

Estructura básica de un programa usando COBLIB

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    EJEMPLO.
   ::::::::::
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

           COPY "COBLIB\HTML-OUT.SEL".

       DATA DIVISION.
       FILE SECTION.

       COPY "COBLIB\HTML-OUT.FD".

       WORKING-STORAGE SECTION.

       COPY "COBLIB\TEMPLATE.WSS".

       01  WEB-DATA.
           02  VARIABLE1               PIC X(01).
           02  VARIABLE2               PIC X(01).

       LINKAGE SECTION.
       COPY "COBLIB\LS-HTTP.FDT".

       PROCEDURE DIVISION USING HTTP-CONTROL HTTP-DATA-INP.
       0000-START.
           MOVE HTTP-DATA-INP  TO WEB-DATA.
           PERFORM TPL-OPEN-FILE.

           MOVE "tpl\template1.html" TO TPL-FILENAME.
           PERFORM TPL-ADD-FILENAME.

           CLOSE HTML-OUT.

           GO TO 999-EXIT.

       COPY "COBLIB\TEMPLATE.FDT".

       999-EXIT.

       EXIT-PROG.
           EXIT PROGRAM.
       END PROGRAM.
```

En este pequeño esqueleto se ve que se usan los archives más importantes de COBLIB. Y lo que se hace es un simple COPY en la sección que se necesita.

Lo interesante empieza en la PROCEDURE DIVISION, en el párrafo 000-START.

Aquí lo primero que hacemos es:
```
MOVE HTTP-DATA-INP  TO WEB-DATA.
```
Esta linea lo que hace es, separar los datos que envia COBHTTPD, y convertirlos a nuestros datos de entrada, en este caso usariamos dos variables de entrada:
1.- VARIABLE1
2.- VARIABLE2

Debe de ser lo primero, para que ya en las instrucciones siguientes podamos hacer uso de estas variables de ser necesario.

En la segunda línea, hacemos:
```
PERFORM TPL-OPEN-FILE.
```
Esta llamada a TPL-OPEN-FLE es la ejecución de un párrafo que esta en TEMPLATE.FDT
(Recuerda que este archivo contiene todos los párrafos o rutinas para trabajar con templates y el envío de información hacia COBHTTPD)

Esta llamada lo que hace es, crear el archivo de resultado que le vamos a aventar a COBHTTPD, el nombre del archivo a enviar viene en HTTP-OUTPUT (Explicado unas paginas atrás).
Entonces en general lo que hace es, saca dicho nombre, se lo asigna a nuestro archivo de trabajo (HTML-OUT) y lo abre y deja abierto para empezar a escribir sobre el.

La tercer cosa que hacemos:
```
MOVE "tpl\template1.html" TO TPL-FILENAME.
PERFORM TPL-ADD-FILENAME.
```
Indicamos que archivo de template vamos a usar para regresar el contenido html.
En este caso indicamos que es el archivo TEMPLATE1.HTML, y que esta dentro del directorio TPL\.

Y con el PERFORM que hacemos a TPL-ADD-FILENAME
Lo que hacemos es escribir el comando hacia COBHTTPD para indicarle el archivo de template a usar.
Siempre tenemos que hacer las dos cosas, primero indicar el archivo y luego ejecutar el párrafo.

Aquí ya podríamos hacer cualquier cosa adicional dependiendo del programa.
Al final, cerramos el archivo:
```
CLOSE HTML-OUT.
```
Y Termina la ejecución del programa.

Preparando la respuesta al cliente.

Una de las primeras cosas que hay que hacer al regresar contenido web, es preparar el archivo de salida para el cliente.

Esto se hace con la siguiente instrucción:

```
PERFORM TPL-OPEN-FILE.
```

La llamada a este párrafo lo que hace es:
1.- Pasar el nombre del archivo que viene en HTTP-OUTPUT a el nombre de HTML-OUT.
2.- Abrir de output el archivo HTML-OUT y dejarlo listo para empezar a escribir.

Al finalizar el proceso de escribir en el archivo de salida siempre debe de cerrarse:

```
CLOSE HTML-OUT.
```

Indicar el archivo de template a utilizar.

COBLIB funciona utilizando archivos de template html.
De esta manera, en lugar de armar todo el contenido html desde dentro de COBOL, simplemente se indica que archivo HTML va a regresar.

El archivo de template es un archivo HTML regular, con algunas marcas especiales, que sirven para que COBOL incruste su propio contenido dentro.

Para indicar el template a usar, usamos la siguiente forma:

```
MOVE "dir\mi-template.html" TO TPL-FILENAME.
PERFORM TPL-ADD-FILENAME.
```

1.- Movemos el nombre del archivo a la variable TPL-FILENAME.
2.- Ejecutamos el párrafo TPL-ADD-FILENAME.

En este párrafo, se encarga de escribir en el archivo de salida HTML-OUT la indicación del archivo de template a usar.
De esta forma, cuando COBHTTPD regrese el contenido, va a regresar el contenido del archivo indicado como si fuera html completo.

Enviar una variable de COBOL hacia HTML

Para pasar una variable desde COBOL hacia el template que se va a regresar se hace de la siguiente manera:

```
MOVE "NOMBRE-VAR" TO VAR-NAME.
MOVE VARIABLE TO VAR-VALUE.
PERFORM TPL-ADD-VAR.
```

1.- Se le pone el nombre de la variable HTML en VAR-NAME.
2.- Se manda el valor de la variable a VAR-VALUE.
3.- Se ejecuta el párrafo TPL-ADD-VAR.

Con esto TPL-ADD-VAR lo que hace es generar el código para insertar este valor en el template HTML que se este usando.

Dentro del template, se puede colocar una variable con el formato:

```
<? NOMBRE-VAR ?>
```

No importa donde se coloque, siempre esa marca va a ser substituida por el valor que le mandemos desde nuestro programa COBOL.
En los ejemplos, esto se puede ver en el programa PRIMER2.CBL, donde se mandan variables al template que se usa y este es PRIMER2.HTML

El nombre de la variable no importa si es mayúsculas o minúsculas, es indistinto.

Variables de entrada y Variables del Servidor

Como ya se menciono en paginas anteriores, todas las variables que necesita el programa deben estar primero registradas en el proyecto.
Y todas se recibirán en el registro HTTP-DATA-INP que esta declarado dentro del archivo LS-HTTP.FDT,

Este archivo se incluye en la LINKAGE SECTION:

```
LINKAGE SECTION.

COPY "COBLIB\LS-HTTP.FDT".
```

Luego, en la PROCEDURE DIVISION, se debe descomponer el valor de HTTP-DATA-INP para poder obtener las variables tal como vienen de los formularios:

```
PROCEDURE DIVISION.
INICIO.
    MOVE HTTP-DATA-INP TO MIS-VARIABLES.
```

En este ejemplo, utilizamos un registro propio del programa que se llama MIS-VARIABLES.
Esta variable es un registro, que debe tener declarada todas las variables con las que vamos a trabajar, y deben de estar en el mismo orden como se declararon en el proyecto en el cfgcobhttpd.exe

En la WORKING-STORAGE SECTION, quedaría de esta manera:

```
01 MIS-VARIABLES.
   02 VARIABLE1        PIC X(10).
   02 VARIABLE2        PIC X(15).
   02 VARIABLE3        PIC S9(12)V99.
```

En conclusión, para obtener los valores de las variables que vienen de los formularios web o incluso desde la barra de dirección, simplemente movemos la variable HTTP-DATA-INP hacia nuestra variable propia, y automáticamente esas variables nuestras se llenan con los datos que vienen desde los formularios.

Hay una serie de variables especiales que es el propio servidor COBHTTPD quien alimenta estos valores, estas variables especiales son:

HTTP-HOST	Es el nombre del dominio o la ip que se tecleo en la barra de direcciones del cliente. Sirve para obtener el dominio desde el que están accesando. En caso de que entraran por la IP, entonces nos mandaria el valor de la IP que pusieron en la barra de direcciones.
HTTP-SERVER-ADDR	Esta es la dirección IP del servidor por la que entro la petición. Suponiendo que nuestro server tuviera varias tarjetas de red conectadas a diferentes proveedores, esta variable nos podría decir por cual IP es por la que esta accesando un cliente.
HTTP-REMOTE-ADDR	Esta es la dirección IP del cliente que esta accesando. Es la dirección remota desde donde están haciendo las peticiones. Serviria para llevar un LOG o registro de operaciones por cada petición.
HTTP-USER-AGENT	Esta variable nos arroja una firma del navegador por el que están accesando, normalmente trae el nombre del navegador, asi como la versión y el sistema operativo del que esta accesando el cliente.

Para obtener estos valores en nuestro programa, solamente hay que registrarlos primero en el proyecto en el programa deseado, y posteriormente declararlas en nuestro registro interno de la WORKING-STORAGE SECTION.
En el ejemplo anterior, si incluyéramos todas estas variables, quedaría de esta forma.

```
01 MIS-VARIABLES.
   02 VARIABLE1           PIC X(10).
   02 VARIABLE2           PIC X(15).
   02 VARIABLE3           PIC S9(12)V99.
   02 HTTP-HOST           PIC X(50).
   02 HTTP-SERVER-ADDR    PIC X(20).
   02 HTTP-REMOTE-ADDR    PIC X(20).
   02 HTTP-USER-AGENT     PIC X(50).
```

Los tamaños de las variables deben de coincidir exactamente con los declarados en el proyecto.

Variables de Sesión

COBHTTPD nos permite manejar sesiones, con el fin de recuperar un valor de una variable la siguiente vez que el usuario accese al mismo programa.

Como es sabido, el protocolo http solo regresa el contenido de una consulta. Una vez que el contenido es regresado al navegador se pierde la conexión entre el servidor y el cliente. De tal manera que la siguiente vez que se realiza la consulta a otro documento se hace una conexión nueva y los datos no se pasan de una consulta a otra. Aun cuando se trate del mismo cliente.

Para evitar este problema COBHTTPD maneja sesiones, las cuales permiten grabar valores y pasarlos de consulta en consulta durante el tiempo que dure la sesión.

Las sesiones siempre se crean y se actualizan en cada consulta. Para grabar un valor en una sesión desde nuestro programa COBOL, debemos declarar lo siguiente:

```
MOVE "VAR-SESION" TO SESS-NAME.
MOVE VALOR-SES TO SESS-VALUE.
PERFORM TPL-ADD-SESSION.
```

Con esto, estamos haciendo lo siguiente:

1.- le indicamos el nombre de la variable de sesión. Esta variable siempre la podremos obtener solo declarándola, no tiene que venir de un formulario o de la url, lo almacenamos y obtenemos el valor cuando lo necesitemos.
2.- Indicamos el valor que deseamos guardar, es un alfanumérico y siempre obtendremos el valor tal cual lo registramos.
3.- Ejecutamos el párrafo TPL-ADD-SESSION, este párrafo hace lo necesario para generar el código hacia COBHTTPD y guardar dicho valor como variable de sesión.

El uso que le podemos dar a una variable de sesión es por ejemplo.

Si el usuario ya se logueo, entonces podemos mandar el numero de usuario a una variable de sesión, y saber asi de que usuario estamos hablando.
Si es un cliente, el que esta haciendo un pedido por la web, podemos mandar el numero de cliente y en cualquier lado, saber que cliente es el que esta haciendo el pedido o navegando en la web.

A mi me gusta mucho utilizar una variable de sesión para saber si el usuario ya hizo login. En los ejemplos, el MENU01 es un programa que hace login, cuando valida que el usuario y el password están bien, crea una variable de sesión que se llama LOGINOK, y le almaceno el valor COB1357.

En esa serie de ejemplos, cuando quieren entrar a las opciones REGISTRO DE CLIENTES o a la de LISTADO DE CLIENTES, estos programas lo primero que hacen es ver que valor tiene la variable de sesión LOGINOK, si no trae COB1357, entonces no hizo login por donde debe, y ejecuta el programa de login, y el registro de clientes no procede.

Lo especial de una variable de sesión, es que nadie puede alterar dicho valor, por que nadie sabe como se llama nuestra variable de sesión, y mucho menos nuestro valor secreto.
Son variables y valores que se controlan desde nuestro programa únicamente.

En el caso de guardar un número de cliente o numero de socio o empleado, debió haberse hecho previamente una autenticación básica, para poder guardar dicho valor.

Para recuperar el valor que hemos grabado en la sesión, debemos de tomar el valor como un parámetro de entrada. Es decir registrarlo en la Configuración de COBHTTPD como variable de entrada de nuestro programa, y declararlo en el registro para poder tener el valor siempre que se entre a nuestro programa.

Pasar valores a controles especiales de un Form

Hay ciertos controles especiales en los que asignarles un valor no es tan sencillo como mandar la variable y acomodarla en el template.
Estos controles del formulario que son especiales son:

Radio	Es un control donde se puede seleccionar entre 2 o más opciones.
Checkbox	Es un control donde se puede marcar o desmarcar el valor.
Select	El control similar en Windows seria un Combo Box, donde hay una lista de opciones y se puede seleccionar una de ellas.

El valor que se le pase, obviamente es un valor que ya esta establecido en el template.
Para asignarle un valor a este tipo de controles, usamos lo siguiente:

```
MOVE "NOMBRE-CTRL" TO FORM-NAME.
MOVE "VAL-CTRL" TO FORM-VALUE.
PERFORM TPL-DATA-FORM.
```

1.- indicamos el nombre del control al que le vamos a mandar el valor.
2.- registramos el valor que vamos a mandar, si es un valor de una variable simplemente se mueve la variable a FORM-VALUE
3.- Se ejecuta el párrafo TPL-DATA-FORM, y este lo que hace es mandar el comando a la salida para COBHTTPD.

Para cualquier tipo de control se usa el mismo método.
El nombre del control, es el que se estableció como como propiedad NAME en el html.

Esto se usa en el ejemplo CTRLS2.CBL y CTRLS2.HTML, ahí se mandan valores a cada uno de estos controles aquí explicados.

Agregar opciones a la lista de un COMBO / SELECT

Un select (COMBO BOX de windows), es una lista de opciones de las cuales podemos seleccionar una.

En la pagina anterior vimos como seleccionar una opción de la lista de un select.
Pero cuando lo que queremos es armar la lista por programación, entonces podemos utilizar lo siguiente:

```
MOVE "OPCION X" TO COMBO-ITEM.
MOVE "001"  TO COMBO-VALUE.
MOVE "Y" TO COMBO-SELECTED
PERFORM TPL-ADD-TO-COMBO.
```

Los valores que se usan son:

COMBO-ITEM	Es la descripción de la opción que se va a agregar al select.
COMBO-VALUE	Es el valor que va a tener dicha opción al seleccionarla. Este valor es el que se mandaria de regreso a COBOL cuando se seleccione y se de enviar.
COMBO-SELECTED	Indica que es la opción seleccionada con una "Y", o en caso de que no sea seleccionada entonces mover "N".

Al final termina con la ejecución del párrafo TPL-ADD-TO-COMBO, el cual se encarga de enviar esta información a COBHTTPD y llenar template.

En los ejemplos, el programa COMBO1.CBL usa estos métodos, y el template que utiliza es COMBO1.HTML

Manejo de bloques repetitivos

En ocasiones es necesario repetir “N” veces una misma información, para armar por ejemplo una lista de renglones, como si fuera una tabla.
Esto es muy común cuando manejamos una página tipo Browse, donde se puede navegar entre registros a modo de renglones.
Es necesario que por cada lectura de un archivo, se escriba un renglón, para que al dibujar el template html en el cliente, este muestre una tabla o browse de registros.

Esto se hace por medio de algo que en COBHTTPD se llaman bloques, donde en el template hay un bloque que se repite.
Este bloque es un “trozo” de código html, que vamos a repetir tantas veces como queramos.
Luego que tenemos el bloque delimitado, tenemos que buscar en donde lo vamos a pegar, el área donde se pega está marcado por la marca paste.

Como declararlo en el template HTML

Dentro de un template html, se vería como algo así:

```
<!--? paste tabla ?-->
<!--? block row1 ?-->
    <tr>
      <td class="renglon1"><? clave ?></td>
      <td class="renglon1"><? nombre ?></td>
    </tr>
<!--? endblock row1 ?-->
<!--? block row2 ?-->
    <tr>
      <td class="renglon2"><? clave ?></td>
      <td class="renglon2"><? nombre ?></td>
    </tr>
<!--? endblock row2 ?-->
```

Primero, la marca **paste tabla**, indica el lugar donde se va a pegar cada bloque, y el nombre a esta ubicación está indicado como **tabla**, pero podemos tener diferentes lugares para pegar diferentes bloques en todo el html, y podemos irlos nombrando como se necesite.

Luego, tenemos el inicio del bloque, que está indicado por la marca **block row1**, donde **row1** es el nombre del bloque.
Y todo el bloque a repetir, está marcado hasta la marca **endblock row1**. Esto quiere decir que todo lo que está entre **block** y **endblock** es el bloque repetitivo.

Si observan, en la parte de abajo hay otro bloque, que se llama row2. Podemos tener tantos bloques como deseemos, cada uno es para una tarea específica.
La única regla, es que no puede haber un bloque dentro de otro bloque.
Aquí se definen dos bloques, pero primero se termina uno, y luego comienza el otro.

También hay que observar que dentro del bloque row1 y row2 hay variables en juego.
Esto quiere decir que antes de pegar un bloque, tenemos que darle valor a dichas variables, para que cuando lo peguemos ya lleve dichas variables y se arme el código como si fuera una lista de renglones.

Como utilizarlo en COBOL

Dentro de nuestro programa COBOL utilizaremos el siguiente formato para usar y repetir un bloque:

```
MOVE "ROW1"  TO BLOCK-NAME
MOVE "TABLA" TO BLOCK-PASTE.
PERFORM TPL-OPEN-BLOCK.
```

Para repetir un bloque básicamente se necesitan dos variables:

1.- BLOCK-NAME, que indica el nombre del block que vamos a pegar o repetir.
2..- BLOCK-PASTE, indica el nombre de la marca donde pegar el block.
Con la llamada al párrafo TPL-OPEN-BLOCK, lo que hace es enviar la petición al cobhttpd de duplicar el bloque en este caso ROW1, y pegarlo en donde esta la marca que se llama TABLA.

Estos nombres son solo para seguir con el ejemplo de cómo se declaro en el template en la página anterior.
Pero bien pueden ser cualquier otro nombre tanto para el BLOCK-NAME como para el BLOCK-PASTE.

Y solo recordando, que si dentro del bloque tenemos variables que se necesitan para rellenar el bloque, es necesario pasarles el valor antes a dichas variables, y esto se logra usando el párrafo TPL-ADD-VAR explicado anteriormente.

En el ejemplo MENU02.CBL se utilizan bloques, y el template donde podemos verlo es el ACT_CLI_BROWSE.HTML.

Envío de información grande usando MEMOS

Un MEMO es una variable grande, que puede tener varias líneas de texto, que no es posible enviarla en un solo renglón usando una simple VAR.

Dentro de un template HTML, la variable con este contenido tipo MEMO, se podría registrar como cualquier variable normal, ejemplo:

```
<? mi-memo ?>
```

Desde COBOL, para mandar el valor a este memo, tendríamos que primero hacer lo siguiente:

```
MOVE "mi-memo" TO MEMO-NAME.
PERFORM TPL-START-MEMO.
```

Con esto se indica el nombre de la variable MEMO que se va a mandar.

Luego tendríamos que agregar el contenido a dicho memo:

```
MOVE "ESTA ES UNA LINEA" TO DAT-MEMO.
PERFORM TPL-DAT-MEMO.
MOVE "ESTA ES OTRA LINEA" TO DAT-MEMO.
PERFORM TPL-DAT-MEMO.
```

Y así repetir cuantas veces sea necesario para mandar el valor completo.

Y terminar cerrando la declaración del memo con lo siguiente:

```
PERFORM TPL-END-MEMO.
```

Con esto se pueden mandar varias líneas para un texto muy largo.
Cada línea puede tener hasta 1024 chars de longitud.
Podríamos usarlo para darle un valor a un TEXTAREA por ejemplo.

Envío de correos desde nuestra página web.

Así cómo es posible generar reportes web usando un COBVIEW interno, también es posible enviar correos electrónicos desde nuestra aplicación web.
Esto se debe gracias a que tiene integrado también una versión especial de COBEMAIL que funciona en conjunto con el COBHTTPD.

También es posible hacer el envio del email utilizando COBEMAIL, y desde el programa COBOL armar el archivo texto especial para el envío del correo, y luego al final hacer el CALL “SYSTEM” USING “COBSENDM.EXE xxxxxxxxxxxxxx”.

No es como que sea mejor una que otra, en realidad es lo mismo.
Lo único es que COBHTTPD ya trae incorporado un COBEMAIL propio, por lo que no se necesita una licencia especial para este.

Para poder enviar correos desde COBHTTPD, es necesario que este configurada toda la parte de Send Mail, en la sección COBAPP del programa Configurador (cfgcobhttpd.exe).

Como declarar el archivo para el correo

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    EJEMPLO.
   ::::::::::
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

           COPY "COBLIB\HTML-OUT.SEL".
           SELECT EMAILDATA
                  ASSIGN TO INPUT-OUTPUT WF-EMAILDATA
                  ORGANIZATION IS LINE SEQUENTIAL
                  ACCESS MODE  IS SEQUENTIAL
                  FILE STATUS  IS STAT-EMAIL.

       DATA DIVISION.
       FILE SECTION.

       COPY "COBLIB\HTML-OUT.FD".
       FD  EMAILDATA
           LABEL RECORDS ARE OMITTED.
       01  LINEA         PIC X(1024).
```

```
       WORKING-STORAGE SECTION.

       COPY "COBLIB\TEMPLATE.WSS".

       COPY "COBLIB\COBEMAIL.WSS".
       77  STAT-EMAIL                PIC X(02).
       77  WF-EMAILDATA              PIC X(150).

       01  WEB-DATA.
           02  VARIABLE1                PIC X(01).
           02  VARIABLE2                PIC X(01).

       LINKAGE SECTION.
       COPY "COBLIB\LS-HTTP.FDT".

       PROCEDURE DIVISION USING HTTP-CONTROL HTTP-DATA-INP.
       0000-START.
           MOVE HTTP-DATA-INP  TO WEB-DATA.
           PERFORM TPL-OPEN-FILE.

           MOVE "tpl\template1.html" TO TPL-FILENAME.
           PERFORM TPL-ADD-FILENAME.

           ::::::::::::

           MOVE WF-EMAILDATA TO EMAIL-FILE.
           PERFORM TPL-SEND-EMAIL.

           CLOSE HTML-OUT.

           GO TO 999-EXIT.

       COPY "COBLIB\TEMPLATE.FDT".

       999-EXIT.

       EXIT-PROG.
           EXIT PROGRAM.
       END PROGRAM.
```

Esta es una estructura muy básica de un programa que aparte de regresar un contenido html con un template, también envía un correo.
Lo que hay que hacer extra es:
1.- Declarar tanto el SELECT como el FD de el archivo de correo, también se declaran las variables del archivo como el Estatus y el nombre variable del archivo.

2.- Se hace un COPY al archivo COBEMAIL.WSS, que contiene todos los registros de comandos que se pueden enviar para el correo, como las cuentas FROM, TO, CC para indicar las cuentas de correo a usar, y otros comandos.

Para indicar que se va a enviar un email al mismo tiempo que la respuesta html hacia el cliente, se usa el siguiente comando:

```
MOVE WF-EMAILDATA TO EMAIL-FILE.
PERFORM TPL-SEND-EMAIL.
```

Primero se mueve el nombre del archivo de correo a la variable EMAIL-FILE.
Y lo segundo es ejecutar el párrafo TPL-SEND-EMAIL, con esto se le indica al COBHTTPD, que aparte de regresar el contenido al cliente, también envie un email Utilizando todos los comandos de EMAIL que van dentro de este archivo indicado en EMAIL-FILE.

Si se quisieran ver ejemplos de cómo armar este archivo, pueden descargarse los ejemplos para COBEMAIL del sitio www.cobtools.com
Y para ver como funcionan todos los registros que tiene COBEMAIL.WSS, pueden revisar el manual para COBEMAIL, que también lo pueden descargar de la misma pagina.

La diferencia principal radica en que, si se va a enviar el correo desde COBEMAIL, entonces hay que hacer el CALL SYSTEM que traen los ejemplos, y si se va a enviar desde COBHTTPD, entonces hay que hacer el PERFORM TPL-SEND-EMAIL como se explica arriba.

En los ejemplos de COBHTTPD, esta el programa EMAIL.CBL, y ahí se puede apreciar cómo es que funciona el envío de correos desde el COBHTTPD.

Cualquier sugerencia pueden enviarla a aortega@cobtools.com.