Cika Electrónica

Tutorial: CTU-006
Título: **Rabbit Web**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	02/02/06	

El presente es un tutorial sobre RabbitWeb, la extensión a Dynamic C que permite presentar páginas web dinámicas de una forma rápida, sin recurrir a CGIs. Cubrimos las principales características, sintaxis y ejemplos de uso. Se recomienda la lectura de información al respecto para un estudio más profundo.

Índice de contenido

Introducción

RabbitWeb es una extensión a Dynamic C, a partir de la versión 8.50. Si bien no se incluye con Dynamic C sino que tiene un costo adicional, su funcionamiento está totalmente integrado al servidor HTTP que sí se incluye con Dynamic C, y elimina la necesidad de escribir CGIs en C, permitiendo al developer una mayor libertad en el diseño de las páginas web.

RabbitWeb está conformado por dos partes fundamentales:

- Un lenguaje tipo script, embebido dentro de *tags* que son interpretadas por el servidor HTTP al momento de servir la página
- Extensiones al lenguaje de Dynamic C, que permiten simplificar el trabajo con variables.

La forma de trabajo en este entorno es:

- ➔ definir las variables que se van a utilizar para ser accedidas mediante el servidor HTTP, ya sea para configuración o para mostrar una determinada información
- ➔ definir los rangos de validez de las variables a modificar, lo cual probablemente ya se haya hecho como parte de la definición de la aplicación
- ➔ definir los grupos de usuarios que van a acceder (o no) a dicha información
- ➔ definir los métodos de autenticación a emplear para cada variable
- ➔ identificar las acciones a realizar cuando existen modificaciones en algunas variables
- ➔ Una vez hecho esto, la presentación de la información se realiza mediante el lenguaje script, en combinación con HTML, lo que se denomina ZHTML

Extensiones a Dynamic C

Al utilizar RabbitWeb, lo cual se indica mediante la siguiente macro:

```
#define USE_RABBITWEB 1
```

se habilitan una serie de extensiones a Dynamic C que nos permiten trabajar de forma más fácil. Por ejemplo, para definir los grupos de usuarios a emplear, se utiliza la directiva *#web_groups*:

```
#web_groups ADMIN_GROUP, MON_GROUP
```

Los nombres empleados se agregan al espacio de nombres conocidos por Dynamic C, por ejemplo, para agregar un usuario al grupo *ADMIN_GROUP* procedemos como si hubiéramos definido el grupo manualmente:

```
        sauth_setusermask(uid, ADMIN_GROUP, NULL);
```

Las variables a ser accedidas vía web deben registrarse, lo cual se realiza aplicando la directiva *#web* a una variable declarada, de la siguiente forma:

```
int variable;

#web variable
```

Al mismo tiempo que se la registra, es posible definir los grupos de usuarios y sus permisos respectivos, así como también una función de evaluación para chequear si el valor ingresado (al modificarla vía web) está dentro del rango permitido, por ejemplo:

```
#web variable (($variable > -10) && ($variable < 10)) groups=ADMIN_GROUP(rw),MON_GROUP(ro)
```

Según vemos, el valor ingresado deberá estar comprendido entre -9 y 9 inclusive, el grupo *ADMIN_GROUP* tiene permiso de lectura y escritura y el grupo *MON_GROUP* tiene sólo permiso de lectura. Cualesquiera otros grupos, no pueden acceder a esta variable. En el caso que se quiera dar un determinado tipo de acceso a todos los grupos, se los puede referir como *all*:

```
#web variable groups=ADMIN_GROUP(rw),all(ro)
```

El signo *$* delante del nombre de la variable indica que nos estamos refiriendo al valor que se intenta ingresar, es decir, el que envía el usuario desde su navegador. Si omitimos este signo, entonces nos estaremos refiriendo al valor que tiene la variable para el resto del sistema, es decir, el último modificado correctamente:

```
#web variable ($variable > variable)
```

De igual modo, se puede especificar el tipo de autenticación deseado:

```
#web variable auth=basic,digest groups=ADMIN_GROUP(rw),all(ro)
```

Las variables a registrar pueden ser de diversos tipos, RabbitWeb interpreta el tipo de la variable, por lo que es posible registrar una estructura y referirse luego a cada uno de sus elementos tanto en la función de chequeo como en el script para su presentación, por ejemplo:

```
#web config (($config.release > 0)&&($config.release < 10))
```

Es posible asignar un texto a mostrar cuando se sale fuera de rango, con una función especial del lenguaje script (*error($variable)*):

```
#web config (($config.data1 <= 9999)?1:WEB_ERROR("muy alto"))
```

También es posible registrar un array, y definir la función de chequeo para cada elemento, de la siguiente forma:

```
int vector[10];
#web vector[@] ((vector[@] > -10) && (vector[@] < 10))
```

Los strings se registran por el nombre del array, y el sistema chequea automáticamente de no exceder la longitud al actualizar:

```
char somestring[25];
#web somestring
```

Cuando en el navegador se envía un formulario (*submit*), éste inicia un requerimiento de una página al servidor HTTP del Rabbit indicando una operación *POST*, y envía los datos. RabbitWeb entonces chequea si hubo cambios y valida acorde a las funciones especificadas, actualizando las variables correspondientes. Si el cambio de una de estas variables implica que se debe realizar una actividad adicional, como por ejemplo un port serie (cerrar y abrir), una dirección IP (convertirla, bajar y subir la interfaz), o simplemente salvar la configuración en flash, podemos indicarlo mediante la directiva *#web_update*:

```
void ejecutame(void)
{
        // tarea relacionada con una modificación de variable
}
#web_update variable ejecutame
```

La función *ejecutame()* se ejecuta entonces cuando RabbitWeb detecta una modificación válida sobre *variable*.

Si deseamos incluir selectores en nuestra página, disponemos de una función que nos permite asignar valores dentro de un entero a los textos del selector, la lista se auto-enumera, y puede ser modificada a voluntad:

```
int lista;
#web lista select( "uno" = 1, "dos", "tres", "cuatro" )
```

También podemos definir botones de selección (*radio buttons*) y casillas para opciones seleccionables (*checkboxes*). Los botones se definen de igual forma que un selector, las casillas son variables enteras.

Los selectores y botones incluyen "rangos de validez", es decir, solamente aceptan los valores que han sido definidos en el programa.

El checkbox es esencialmente un "sí/no", y RabbitWeb acepta cualquier valor. Se recomienda que nuestro programa no dependa del valor *1* para considerarlo habilitado, sino que considere cualquier valor diferente de *0* como "*sí*"; de todos modos puede definirse un rango de validez si se desea.

Lenguaje script

De forma similar a como SHTML incluye tags que son interpretados por el servidor al momento de servir la página (SSI: Server Side Includes), ZHTML permite disponer de un simple y poderoso lenguaje script dentro de la página, que será interpretado por servidor HTTP de Rabbit al momento de servir la página.

Los tags se incluyen dentro de un envoltorio:

```
        <?z   ?>
```

Dentro del mismo, una sentencia por línea, se coloca el script; por ejemplo, el código a continuación hace que el servidor muestre el valor de la variable variable:

```
<?z print($variable) ?>
```

Cada variable a que se haga referencia, deberá haber sido registrada en el programa principal, como viéramos en el apartado anterior. Por claridad, utilizaremos los mismos nombres de variables que definiéramos. Cada variable referida en el script deberá estar precedida por un signo *$* o un signo @. El signo *$* indica que estamos haciendo referencia al valor que se intenta introducir al enviar la página; el signo @ indica que nos referimos al valor que tiene la variable en el programa principal.

La sintaxis es similar a C y la gramática es simple, permitiendo ejecución condicional mediante el uso de la cláusula *if()*, aunque sin su *else* complementario, lo que se resuelve invirtiendo la pregunta (la cual debe realizarse nuevamente mediante el operador de negación: *!* (signo de admiración):

```
<?z if($variable==1) { ?>
        acción
<?z } ?>
```

La acción encerrada entre llaves puede ser código HTML, el cual forma páginas diferentes dependiendo de lo que considere el script, o parte del script, incluyendo más condicionales anidadas. Debido a que esto implica un consumo de memoria, existe un límite a la cantidad de veces que se puede anidar, que se define por programa mediante la macro *RWEB_ZHTML_MAXBLOCKS*, que por defecto es 4. Esto aplica también e incluye[1] bloques iterativos, representados mediante la cláusula *for(; ;)*. Las variables utilizadas dentro de un *for* son válidas en el entorno de ejecución del script:

```
<?z for($A=0; $A<10;$A++) { ?>
        acción
<?z } ?>
```

El resto de la operación se realiza mediante funciones que permiten resolver situaciones comunes de un modo simple y efectivo.
Para conocer la cantidad de elementos de un array o selector se utiliza la función count(), por ejemplo:

```
<?z for ($A = 0; $A < count($lista); $A++){ ?>
        acción
<?z } ?>
```

En un array, debemos especificar además a qué dimensión nos referimos, por ejemplo, un array unidimensional (vector):

```
<?z for ($A = 0; $A < count($vector,0); $A++){ ?>
        acción para vector[$A]
<?z } ?>
```

Para una matriz, el segundo parámetro será 0 ó 1, según a qué dimensión nos queramos referir.

Para generar un selector:

```
<SELECT NAME="lista">
<?z print_select($lista) ?>
</SELECT><br>
```

Si se desea reemplazar los nombres o agregar algo, puede generarse manualmente con la función *print_opt()*. La función *selected()* permite determinar si la opción corriente es la que corresponde al valor actual de la variable en el programa principal:

```
<?z for ($A = 0; $A < count($lista); $A++){ ?>
        <SELECT NAME="lista">
        <OPTION
        <?z if (selected($lista, $A) ) { ?>
                SELECTED
        <?z } ?>
        > <?z print_opt($lista, $A) ?>
        </SELECTED>
<?z } ?>
```

Para generar una lista de botones:

1 La cantidad total de bloques iterativos y condicionales que puede ser anidada es RWEB_ZHTML_MAXBLOCKS

```
<?z for ($A = 0; $A < count($lista); $A++){ ?>
        <INPUT TYPE="radio" NAME="lista"  OPTION
        <?z if (selected($lista, $A) ) { ?>
                CHECKED
        <?z } ?>
        VALUE="<?z print_opt($lista, $A) ?>"> <?z print_opt($lista, $A) ?>
<?z } ?>
```

Para conocer si la página está siendo mostrada como resultado de una petición (GET) o una introducción de valores (POST):

```
<?z if(update()) { ?>
        se trata de un POST
<?z } ?>
<?z if(!update()) { ?>
        se trata de un GET
<?z } ?>
```

Para saber si hay algún error en la información enviada:

```
<?z if(error()) { ?>
        hay errores
<?z } ?>
```

Para determinar si el error corresponde a una variable en particular:

```
<?z if(error($variable)) { ?>
        hay un error en el valor enviado de variable
<?z } ?>
```

Para mostrar el mensaje de error asociado a una variable en particular:

```
<?z print(error($variable)) ?>
```

En el caso particular en que trabajemos con arrays de enteros, los corchetes son caracteres no admitidos dentro del nombre de una variable en HTTP. Para resolver esta situación, se incorpora una función adicional, *varname()*:

```
<INPUT TYPE="text" NAME="varname($variable[$indice])" >
```

Para una descripción detallada y precisa de la gramática y cada una de las funciones y extensiones, se sugiere consultar el manual de RabbitWeb.

Ejemplo

El ejemplo a continuación es una forma simple de realizar un cambio de configuración mediante una página web, con RabbitWeb.

En primer lugar, indicamos que estamos utilizando RabbitWeb:

```
#define USE_RABBITWEB     1
```

lo cual debe hacerse antes de incluir la library del web server.
Luego, definimos los dos grupos de usuarios:

```
#web_groups ADMIN_GROUP
#web_groups MON_GROUP
```

A continuación, registramos las variables a utilizar, las cuales ya deben estar declaradas y deben ser globales. Por comodidad, registramos la totalidad de la estructura, pero podríamos registrar cada uno de sus elementos de forma individual. Si registramos la estructura, es más simple el programa, pero pasamos la complejidad al script (ZHTML):

```
#web config auth=basic,digest groups=ADMIN_GROUP(rw),MON_GROUP(ro)
```

Con la definición anterior registramos la variable *config* (una estructura), con permiso de lectura y escritura para el grupo *ADMIN_GROUP*, permiso de lectura solamente para el grupo *MON_GROUP*, y la autenticación del usuario podrá hacerse mediante los métodos *basic* o *digest*.

Cuando una variable sea actualizada (cambio de configuración), RabbitWeb lo detectará y ejecutará una función. Definimos esa función y la registramos:

```
void saveconfig(void)
{
        writeUserBlock(CONFIG_OFFSET,&config,sizeof(Configureiyon));
        printf("Saved to flash\n");
}

#web_update config saveconfig
```

Una ventaja que tiene el uso de RabbitWeb, es que verifica si realmente hubo cambios, es decir, si lo que se envía es igual a lo que existía, no llama a esta función porque no hay cambios.

Los tipos MIME los definimos como siempre, pero esta vez incluiremos la extensión *zhtml* para el script RabbitWeb, y le asignamos el correspondiente handler:

```
SSPEC_MIMETABLE_START
        SSPEC_MIME_FUNC(".zhtml", "text/html", zhtml_handler),
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".gif", "image/gif"),
SSPEC_MIMETABLE_END
```

El directorio del server lo definimos como siempre, protegiendo la página de configuración:

```
SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/", index_html),
        SSPEC_RESOURCE_XMEMFILE("/index.shtml", index_html),
        SSPEC_RESOURCE_P_XMEMFILE("/setup.zhtml",setup_html,"mi equipo",ADMIN_GROUP|MON_GROUP,0,SERVER_HTTP,
    SERVER_AUTH_BASIC | SERVER_AUTH_DIGEST),
        SSPEC_RESOURCE_XMEMFILE("/rabbit1.gif", rabbit1_gif),
SSPEC_RESOURCETABLE_END
```

Los usuarios se generan en el programa principal.

Finalmente, el resto de la tarea la desarrollamos sobre la página *ZHTML*, la cual incluye un script que nos permite que se ejecuten ciertas cosas y se modifique el código entregado, de una forma algo más simple que si utilizáramos SSI. De esta manera, podemos utilizar la misma página tanto para configurar como para mostrar resultados y errores. En nuestro caso, dado que Rabbit Web revisa si hay o no cambios, y hasta enclava los valores numéricos al máximo del tipo empleado, no tendremos errores en esta página. En un caso real, en el que los valores a configurar tienen un rango de validez, es posible mostrar una interfaz muy amigable con mucho detalle. A continuación, veremos el script *ZHTML*:

```
<html>
<head><title>Config</title></head>
<body bgcolor="#FFFFFF" link="#009966" vlink="#FFCC00" alink="#006666" topmargin="0" leftmargin="0"
marginwidth="0" marginheight="0">
<H1>Configuraci&oacute;n</H1>
<form ACTION="setup.zhtml" METHOD="POST">
<?z if(updating()) { ?>
    <?z if(!error()) { ?>
        <hr><H2><FONT COLOR="#0000FF">Configuración actualizada</FONT></H2><hr>
    <?z } ?>
    <?z if(error()) { ?>
        <hr><H3><FONT COLOR="#FF0000">Revise los errores indicados en rojo</FONT></H3><hr>
    <?z } ?>
<?z } ?>
<table>
<tr><td>
        <?z if(error($config.release)) { ?>
                <FONT COLOR="#FF0000">
        <?z } ?>
Revisi&oacute;n
        <?z if(error($config.release)) { ?>
                </FONT>
        <?z } ?>
<td><input TYPE="TEXT" NAME="config.release" SIZE=5 VALUE=<?z print($config.release) ?>>

<tr><td>
Descripci&oacute;n
<td><input TYPE="TEXT" NAME="config.name" SIZE=64 VALUE="<?z print($config.name) ?>">

<tr><td>
        <?z if(error($config.data1)) { ?>
                <FONT COLOR="#FF0000">
        <?z } ?>
Dato
```

```
        <?z if(error($config.data1)) { ?>
                </FONT>
        <?z } ?>
<td><input TYPE="TEXT" NAME="config.data1" SIZE=5 VALUE=<?z print($config.data1) ?>>
        <?z if(error($config.data1)) { ?>
                <FONT COLOR="#FF0000">
                <?z print(error($config.data1)) ?>
                </FONT>
        <?z } ?>
</table>
<?z if(auth($config.release,"rw")) { ?>
        <input TYPE="SUBMIT" VALUE="Cambiar"></form>
<?z } ?>
</body>
</html>
```

Como vemos, es mayormente similar a un SHTML, el script ZHTML está embebido entre *<?z ?>*, y con un lenguaje muy simple es posible lograr páginas dinámicas muy complejas.

- Si estamos actualizando la página, es decir, el pedido por el cual el server en el Rabbit está mostrando esta página corresponde a un POST ocasionado por la presión del botón Cambiar, y si no hay ningún error, informamos que la configuración fue actualizada.
- En cada campo a configurar, utilizamos el nombre del elemento dentro de la estructura, como haríamos en cualquier programa C.
- Si el usuario que está viendo la página no tiene permiso de escritura, no mostramos el botón *Cambiar*.

A continuación, veremos el listado completo del programa:

```
#class auto

#define USE_RABBITWEB     1

#define CONF_TXTSZ 64

typedef struct {
        int      release;
        char     name[CONF_TXTSZ];
        int      data1;
} Configureiyon;

const static Configureiyon info_defaults = {
1,"Esta es la primera versi&oacute;n",83
};

#web_groups ADMIN_GROUP
#web_groups MON_GROUP

#define CONFIG_OFFSET    (4096*GetIDBlockSize()-0x800)

Configureiyon config;

#web config auth=basic,digest groups=ADMIN_GROUP(rw),MON_GROUP(ro)
#web config (($config.release > 0)&&($config.release < 10))
#web config (($config.data1 <= 9999)?1:WEB_ERROR("muy alto"))
#web config (($config.data1 >= -9999)?1:WEB_ERROR("muy bajo"))

void saveconfig(void)
{
        writeUserBlock(CONFIG_OFFSET,&config,sizeof(Configureiyon));
        printf("Saved to flash\n");
}

#web_update config saveconfig

#define TCPCONFIG 0
#define USE_ETHERNET               1


#define MY_IP_ADDRESS   "192.168.1.54"
#define MY_NETMASK      "255.255.255.0"

#define MY_GATEWAY        "192.168.1.1"


#use "dcrtcp.lib"
#use "http.lib"

#ximport "index.html"   index_html
#ximport "rabbit1.gif"    rabbit1_gif
```

```
#ximport "setup.zhtml"  setup_html

SSPEC_MIMETABLE_START
        SSPEC_MIME_FUNC(".zhtml", "text/html", zhtml_handler),
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".gif", "image/gif"),
SSPEC_MIMETABLE_END


// directorio del server, funciones y variables de SSI
SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/", index_html),
        SSPEC_RESOURCE_XMEMFILE("/index.shtml", index_html),
        SSPEC_RESOURCE_P_XMEMFILE("/setup.zhtml",setup_html,"mi equipo",ADMIN_GROUP|MON_GROUP,0,SERVER_HTTP,
    SERVER_AUTH_BASIC | SERVER_AUTH_DIGEST),
        SSPEC_RESOURCE_XMEMFILE("/rabbit1.gif", rabbit1_gif),
SSPEC_RESOURCETABLE_END


void main()
{
int uid;

        // Load from flash
        if(!BitRdPortI(PBDR,2)) {                   // Reset to defaults
                memcpy(&config,&info_defaults,sizeof(Configureiyon));
                saveconfig();     // save to flash
        }
        else
                readUserBlock(&config,CONFIG_OFFSET,sizeof(Configureiyon));


        if((uid = sauth_adduser("admin", "istrador", SERVER_HTTP))>=0){
                sauth_setusermask(uid, ADMIN_GROUP, NULL);
        }
        if((uid = sauth_adduser("mon", "itor", SERVER_HTTP))>=0){
                sauth_setusermask(uid, MON_GROUP, NULL);
        }

        sock_init();
        http_init();
        tcp_reserveport(80);
        while (1) {
                http_handler();
              // aplicación
        }
}
```

La siguiente es una imagen de RabbitWeb en acción: