
 <b>CONTINEA</b> <small>Microprocesamiento modular + Conectividad</small>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 1 de 17

Revisión	Fecha	Comentario	Autor
0	18/12/2008	Esta nota está dividida en dos secciones: <ul style="list-style-type: none"> <li>• La parte 1 es introductoria para los temas abordados y para la descripción básica del proyecto.</li> <li>• La parte 2, es una descripción mucho más detallada de los módulos que componen la aplicación.</li> </ul>	Ulises Bigliati

## ÍNDICE

<b>PARTE 1</b> .....	<b>2</b>
<b>Introducción</b> .....	<b>2</b>
<b>Incorporando JavaScript a la mezcla</b> .....	<b>2</b>
<b>Que sea con estilo</b> .....	<b>2</b>
<b>La receta completa</b> .....	<b>2</b>
<b>Objetivos</b> .....	<b>3</b>
<b>Implementación</b> .....	<b>3</b>
<b>Descripción del software involucrado</b> .....	<b>4</b>
Estructura de archivos del proyecto.....	4
Archivos del proyecto.....	4
Archivos fuente del proyecto.....	5
Recursos Web del proyecto.....	5
<b>Nota de fin de sección</b> .....	<b>5</b>
 <b>PARTE 2</b> .....	 <b>6</b>
<b>Módulos de software del proyecto</b> .....	<b>6</b>
Módulo config.lib.....	6
Módulo datetime.lib.....	7
Módulo iweb.lib.....	8
Módulo principal.....	11
<b>Los recursos Web del proyecto</b> .....	<b>12</b>
Página principal (index.zhtml).....	12
Página de configuraciones (setup.zhtml).....	13
Hoja de estilos (style.css).....	16
 <b>REFERENCIAS</b> .....	 <b>17</b>

 <b>CONTINEA</b> <small>Microprocesamiento modular + Conectividad</small>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 2 de 17

## Parte 1

### Introducción

Al desarrollador de aplicaciones embebidas que trabaja con el entorno integrado Dynamic C probablemente se le presente en algún momento la ocasión de realizar algún proyecto con la particularidad de incorporar funcionalidad Web en su diseño.

A partir de las versiones 9.60 y 10.21 el Dynamic C puede descargarse gratuitamente desde el sitio Web del fabricante; el módulo de software RabbitWeb, es distribuido en forma integrada y también gratuita junto con el IDE de Rabbit.

RabbitWeb<sup>1</sup> es básicamente un lenguaje de script que corre del lado del servidor, que se integra al http server de Rabbit y que junto a una serie de extensiones en cuanto a directivas de compilador, macros y otras herramientas proporciona un marco de trabajo muy amigable para la implementación de aplicaciones embebidas basadas en Web sin la necesidad utilizar los mecanismos CGI.

### Incorporando JavaScript a la mezcla

Sin embargo, y a pesar de las bondades del módulo RabbitWeb, cuando la interfaz de usuario que tenemos que construir se va haciendo más compleja, es muy conveniente valerse de otro lenguaje de script para ejecutar ciertas tareas, pero esta vez del lado del cliente, es decir que ciertos procesos serían ejecutados por el propio navegador en la computadora del cliente. Este lenguaje es por supuesto JavaScript<sup>2</sup> y las tareas a realizar con este lenguaje son típicamente:

- Funciones de validación utilizadas en forma previa al envío de datos de formularios HTML (para que permitir que el usuario envíe lo que no debe a nuestro sistema embebido si podemos evitarlo antes)
- Manejo de timers: utilizados para ejecutar tareas sincrónicas, tales como el refresco periódico de página, despliegue de relojes en pantalla, y lo que fuera necesario.
- Funciones de presentación que pudieran ser necesarias para adaptar y modificar el contenido HTML de la página y/o reaccionar con la interacción del usuario.
- Creación de menús, etc., etc.

Los ítems listados arriba son solo algunas de las posibilidades que ofrece el uso del lenguaje, el programador seguramente sabrá encontrar muchas otras.

### Que sea con estilo

Por último, no menos importante que la funcionalidad provista y la facilidad de uso es la presentación estética de la interfaz Web que se construya. Mucho más importante aún, desde la óptica del desarrollador es poder realizar el mantenimiento de la estética de la interfaz con el menor esfuerzo posible. En este aspecto, es muy conveniente la incorporación de las hojas de estilo en cascada, archivos que podrán encontrarse por ahí con la extensión .CSS (de cascade style sheet) que cuando son correctamente utilizados, proporcionan una forma muy cómoda de aplicar y modificar estilos a nuestras páginas HTML, separando de esta forma la estructura de presentación estética, de la estructura HTML funcional, esto estaría de acuerdo con una arquitectura dividida en capas, tal como les gusta decir a algunos.

### La receta completa

Desarrollar un proyecto que requiera o justifique la utilización de los elementos que fuimos mencionando, implica manipular y combinar diferentes recursos y/o tipos de codificación:

- El código en **Dynamic C + RabbitWeb** que soportará las funcionalidades de interfaz Web
- Una mezcla de **HTML**, **ZHTML** (el lenguaje scripting de Rabbit) y **JavaScript** en las páginas.
- El complemento de las hojas de estilo **CSS**, que es invocado desde las páginas ZHTML.

<sup>1</sup> Puede consultarse un tutorial en castellano sobre RabbitWeb en la sección de "Tutorials" sitio web de Cika Electrónica :[http://www.cika.com.ar/soporte/Tutorials/CTU-006\\_RabbitWeb.pdf](http://www.cika.com.ar/soporte/Tutorials/CTU-006_RabbitWeb.pdf) o referirse al manual del fabricante en <http://www.rabbit.com/documentation/docs/manuals/TCPIP/UsersManualV2/index.html> (sección 5).

<sup>2</sup> Probablemente una de las mejores referencias de la Web para empezar a comprender este lenguaje: <http://www.w3schools.com/JS/default.asp>

Por otra parte, el kit de herramientas a utilizar para el desarrollo de un sistema basado en Web tal como lo fuimos describiendo podría ser:

1. El **entorno de desarrollo Dynamic C**, para la programación de nuestro sistema dedicado.
2. Un **editor HTML que interprete la sintaxis de Javascript**, para el diseño de la interfaz Web que estará embebida en nuestro sistema.
3. El **navegador** de nuestra preferencia, para visualización, prueba y debug de la interfaz Web. Se pueden diferenciar dos instancias de prueba:
  - o Durante la edición de la página, cuando todo transcurre en la computadora del programador, esto es una prueba mayormente estática, mediante cual pueden comprobarse aspectos de presentación y de ejecución de algunas rutinas de JavaScript.
  - o Cuando la página es provista por el servidor Web del módulo Rabbit, es decir, cuando la interfaz Web está en pleno funcionamiento y la prueba entonces es totalmente dinámica, existiendo interacción con nuestro sistema en desarrollo.

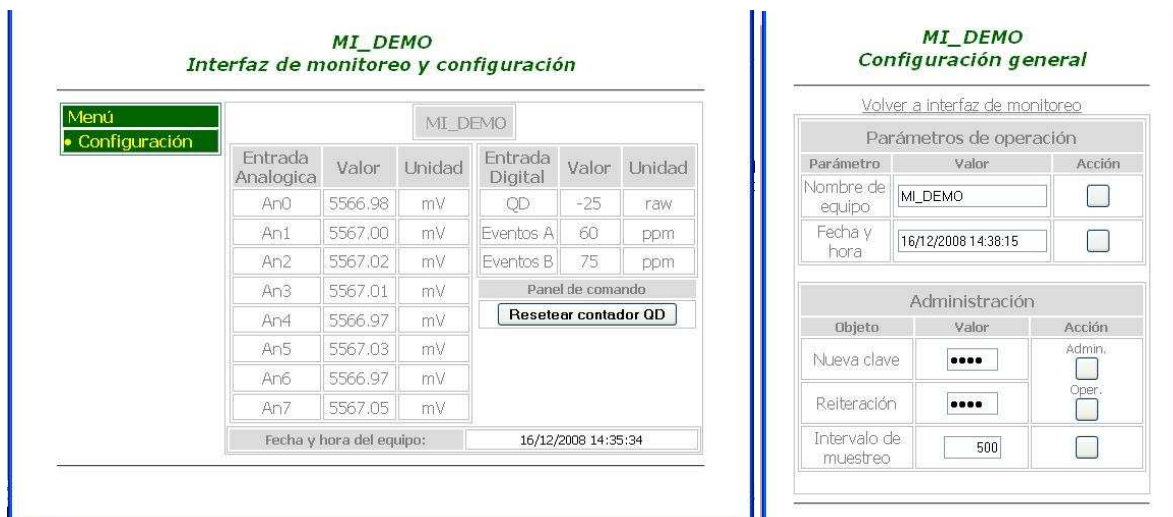
## Objetivos

La propuesta es compartir un pequeño ejemplo de desarrollo de una interfaz Web de control y monitoreo. No interesa mostrar las posibilidades no solo del módulo RabbitWeb, sino de cómo esta herramienta puede combinarse con otros estándares, tales como JavaScript y/o CSS para potenciar así varios aspectos del desarrollo basado en Web.

## Implementación

En esta ocasión, no existen aspectos para considerar vinculados al hardware. Solo cabría destacar que como esta nota es producto de un desarrollo realizado sobre un RCM4300, las características que son propias de los periféricos de este módulo fueron simuladas con fines de generalizar la demostración para que pueda ser probada con otros módulos.

La implementación podría ser realizada entonces en cualquier módulo Rabbit con Ethernet, sin embargo hasta el momento solo hemos comprobado el funcionamiento de esta demo sobre el módulo RCM4300. A continuación podemos ver las imágenes de la interfaz Web generada: a la izquierda, la pantalla principal; a la derecha, la interfaz de configuraciones.



**MI\_DEMO**  
*Interfaz de monitoreo y configuración*

Entrada Analógica			Entrada Digital		
Valor	Unidad		Valor	Unidad	
An0	5566.98	mV	QD	-25	raw
An1	5567.00	mV	Eventos A	60	ppm
An2	5567.02	mV	Eventos B	75	ppm
An3	5567.01	mV	Panel de comando		
An4	5566.97	mV	Resetear contador QD		
An5	5567.03	mV			
An6	5566.97	mV			
An7	5567.05	mV			
Fecha y hora del equipo:		16/12/2008 14:35:34			

**MI\_DEMO**  
*Configuración general*

[Volver a interfaz de monitoreo](#)

Parámetros de operación		
Parámetro	Valor	Acción
Nombre de equipo	MI_DEMO	<input type="checkbox"/>
Fecha y hora	16/12/2008 14:38:15	<input type="checkbox"/>


Administración		
Objeto	Valor	Acción
Nueva clave	••••	Admin. <input type="checkbox"/>
Reiteración	••••	Oper. <input type="checkbox"/>
Intervalo de muestreo	500	<input type="checkbox"/>

Conceptualmente se han definido tres perfiles de usuario para acceder a ellas:

Invitados, operadores, y administradores. Cada uno de ellos obtiene un nivel de acceso diferente, según se ha predeterminado en el código del programa.

Se ha creado además un usuario perteneciente a cada uno de los grupos con los nombres y claves que se listan a continuación:

- invitado: (sin contraseña)
- operador: "1234"
- admin: "6666"

	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 4 de 17

## Descripción del software involucrado

El programa de demostración está escrito en DC10.40 y se trata de la simulación de un sistema de adquisición de datos. La idea representada es la realización de una interfaz de monitoreo Web donde pueden observarse las mediciones (simuladas) de los canales del conversor analógico-digital, del detector de cuadratura y de las entradas de captura de eventos del módulo Rabbit. Adicionalmente se muestran la fecha y hora que esté vigente en el equipo y si el usuario posee los permisos necesarios (es *administrador* u *operador*) también tiene la posibilidad de resetear el contador correspondiente al detector de cuadratura.

Como una segunda y última sección de esta interfaz se incluye una página de configuraciones desde la cual el usuario logueado como *operador* puede modificar la fecha y hora del equipo y también el nombre de este.

Si el usuario ingresa como *administrador* podrá además, cambiar las contraseñas de cualquiera de los dos roles (*operador* / *admin*) y también podrá modificar un parámetro adicional que es el intervalo de muestreo, cuyo sentido sería mucho mayor en una aplicación real.

## Estructura de archivos del proyecto

Físicamente esta demostración está organizada según puede apreciarse en el siguiente esquema. Esta es una organización típica para cualquiera de nuestros proyectos en DC, sin embargo no deja de ser una cuestión de preferencia personal en cuanto a la forma de organización de los archivos de un proyecto.

```

myProjects__CoAN-007
|
|__CoAN-007.c
|__CoAN-007.dcp
|__lib.dir
|__myLibs
|
|   |__iweb.lib
|   |__datetime.lib
|   |__config.lib
|
|__iWeb
|   |__index.zhtml
|   |__setup.zhtml
|   |__style.css


```

A continuación se explicará la naturaleza y funciones específicas de cada uno de los elementos del proyecto, para una mejor comprensión se separarán estos elementos en dos categorías:

### Archivos del proyecto

Los archivos de proyecto son los nombrados a continuación: **myProjects**, **CoAN-007**, **myLibs**, **lib.dir**, y **CoAN-007.dcp**. No se agregarán comentarios al respecto de estos ya que su uso no difiere de lo ya expuesto en notas anteriores (CoAN-006, CoAN-005). Sin embargo, sí haremos un comentario sobre el directorio **iWeb** que es propio de este proyecto en particular:

Este es un directorio creado para alojar los recursos que constituyen físicamente la interfaz web del sistema, es decir, básicamente las páginas ZHTML. Pero para nuestro ejemplo, vemos que también existe el archivo **style.css**, que es la hoja de estilo que aplicamos a nuestras páginas. En este directorio, también depositaríamos los archivos de imagen si existieran y todo recurso relaciona con nuestra interfaz web, como podría ser, por ejemplo un archivo de scripts externo ".js".

	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 5 de 17

## Archivos fuente del proyecto

- **iweb.lib:**  
Se creó este módulo para encapsular todas las funcionalidades propias de la interfaz Web, en esta aplicación en particular constituye el módulo central.
- **datetime.lib:**  
Contiene utilidades de acceso al RTC incluido en todos los módulos Rabbit, las funciones encapsuladas en este módulo permiten por un lado leer y presentar la hora en el siguiente formato: `dd/mm/aaaa hh:mm:ss` y por otro lado permiten poner en hora el reloj a partir de un string de igual formato.
- **config.lib:**  
Concentramos en este módulo todos los objetos que definen la estructura de configuraciones no volátiles que será alojada en el *userblock* (memoria flash del módulo Rabbit). Esta library también contiene todas las funciones para una cómoda grabación y lectura de los datos que aloja.
- **CoAN-007.c:**  
Archivo de programa de la aplicación, que básicamente consta de un loop principal desde el cual se llaman las distintas rutinas para la simulación de los valores que luego pueden ser leídos desde la interfaz Web.


## Recursos Web del proyecto

- **index.zhtml**  
Página web que incluye el código HTML, ZHTML y Javascript necesarios para construir la página principal de la interfaz.
- **setup.zhtml**  
Página compuesta de igual forma que la anterior, para conformar la sección de configuraciones de la aplicación.
- **style.css**  
Hoja de estilos en cascada (Cascade style sheet) que se aplica en forma centralizada a las páginas ZHTML arriba mencionadas para dotarlas de la presentación estética deseada.

## Nota de fin de sección

Si el lector tuvo la paciencia y amabilidad de llegar hasta acá, se merece la aclaración de que hasta este punto, la intención ha sido informar sobre las técnicas que pueden ser aplicadas a sus proyectos con módulos Rabbit y Dynamic C cuando requieran de interfaces Web complejas.

Como sabemos que ver en funcionamiento un software de demostración puede valer más que mil líneas de comentario, avisamos que a estas alturas podría ser conveniente intentar correr el software de demostración suministrado con esta nota, ya que a continuación, en la parte 2 de este documento, se intenta explicar con bastante detalle las técnicas utilizadas. Para esto utilizamos comentarios adicionales realizados sobre el código que en algunas ocasiones en pro de la claridad, ha sido resumido.

 <b>CONTINEA</b> <small>Microprocesamiento modular + Conectividad</small>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 6 de 17

## Parte 2

### Módulos de software del proyecto

Vamos a pasar ahora, a describir el funcionamiento de los diferentes bloques que integran el proyecto. Por supuesto, conforme al objetivo que se ha declarado para esta nota, nos concentraremos mayormente en los aspectos relacionados con la interfaz Web del sistema.

#### Módulo config.lib

Este módulo está encargado de proveer todos los servicios relacionados con el alojamiento y mantenimiento de la estructura de configuraciones en la memoria flash, más concretamente en la zona de memoria flash designada por Rabbit como *user block*. La estructura de configuración de nuestro sistema presenta esta forma:

```
struct config_st{
    unsigned int sample_interval; char
    password[PASSMAXNUM][PASSMAXLEN+1];
    char name[NAMEMAXLEN+1];
};
```

Y la mantenemos espejada en RAM, según lo define esta declaración:

```
static struct config_st config;
```

Si urgamos un poco mas en este módulo, vemos que hay directivas de compilador como esta:

```
#define CALIBMEMSIZE 2048
```

Quieren decir que reservamos un espacio de 2048 bytes del User Block para alojar allí las constantes de calibración que hubiera tenido nuestro sistema si fuera real y no un pobre simulacro como este.

A continuación vemos como con las directivas

```
#define CONFIGOFFSET          CALIBMEMSIZE
#define CONFIGSIZE           sizeof(config)
```

definimos el offset para la estructura y el tamaño. Esto nos será útil y cómodo para definir los elementos internos de la estructura con macros como las que pueden verse a continuación dentro del módulo que en general tienen esta forma:

```
#define SAMPLEINTERVALOFFSET
CALIBMEMSIZE + (unsigned int)((unsigned long)&SAMPLEINTERVAL) - (unsigned long)&config))

#define SAMPLEINTERVALSIZE  sizeof(SAMPLEINTERVAL)
```


Siendo "*sampleinterval*" definido así unas líneas atrás en el módulo que estamos describiendo:

```
#define SAMPLEINTERVAL          config.sample_interval
```

La intención de estas macros es definir el offset y el tamaño de cada elemento interno de la estructura, pero de una forma genérica y fácil de mantener en el futuro.

Antes de que finalicen las macro vemos estas líneas:

```
#define FIRSTTIMEMARKOFFSET    CONFIGOFFSET + CONFIGSIZE
#define FIRSTTIMEMARKSIZE     2
#define NOFIRSTTIMEMARK       0xAAAA
```

 <b>CONTINEA</b> <small>Microprocesamiento modular + Conectividad</small>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 7 de 17

Que se utilizan para detectar la ocasión singular de que el inicio del sistema es el primer inicio de toda la vida del mismo, y así forzar una configuración de fábrica. La función encargada de esto es la función de inicialización de la configuración, que nos muestra la siguiente anatomía:

```
void cfgInit()
{
    if ( cfgIsFirstTime() )
    {
        cfgSetDefault();
        cfgMarkFirstTime();
    }
    else
        cfgReadAll();
}
```

El resto de las funciones provista por este módulo se presentan a continuación y pueden agruparse de la siguiente forma, las que se destinan a los elementos particulares de la estructura:

```
void cfgSavePassword(char * newpass, unsigned char index);
void cfgSaveName(char * newname);
void cfgSaveSampleInterval(unsigned int interval);
```

Y las que cumplen funciones de administración general de la estructura de configuración:

```
void cfgReadAll();
void cfgWriteAll();
void cfgSetDefault();
void cfgMarkFirstTime();
int cfgIsFirstTime();
void cfgInit();
```

Como ejemplo de la labor de una de las funciones de grabación veamos:

```
void cfgSaveSampleInterval(unsigned int interval)
{
    SAMPLEINTERVAL = interval;
    writeUserBlock( SAMPLEINTERVALOFFSET, &SAMPLEINTERVAL, SAMPLEINTERVALSIZE);
}
```

Donde en la primer línea “grabamos” en la estructura alojada en RAM, y a continuación utilizamos la función de librería “writeUserBlock” mediante la cual grabamos en flash, aprovechando las macros que tanto trabajo nos dieron al principio.

El resto de las funciones actúan de forma muy similar a la descrita, por lo tanto no las examinaremos.

### Módulo **datetime.lib**


Esta library no requiere demasiada explicación ya que se reduce a proporcionarle al sistema la siguiente interfaz:

```
void set_time(char *pNewDateTime);
void get_time(char * strdate, char * strtime);
```

La función `set_time` toma del buffer de caracteres apuntado por el argumento un string con la forma: “dd/mm/aaaa hh:mm:ss” y obtiene de allí la nueva fecha y hora desde la cual podrá configurará el RTC del sistema mediante las función `tm_wr(&rtc)` de la library de RTC de Rabbit.

La función `get_time` realiza el proceso inverso, escribiendo en el buffer de caracteres apuntado por el argumento `*strdate` un string con la forma: “dd/mm/aaaa” Y a continuación, escribe un string con el formato “hh:mm:ss” en el buffer de caracteres apuntado por el argumento `*strtime`.

Esta función se vale a su vez de la función `tm_rd(&rtc)`; presente en las libraries de RTC de Rabbit.

 <b>CONTINEA</b> <small>Microprocesamiento modular + Conectividad</small>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 8 de 17

## Módulo iweb.lib

El módulo `iweb.lib` encarna toda la funcionalidad que pensamos ofrecer mediante la interfaz Web. No obstante eso, solo presenta una función accesible desde afuera del módulo:

```
void iweb_init();
```

Esta simplemente registra para el servidor Web a los usuarios vinculándolos a los grupos que hayamos definido junto con su password correspondiente y en forma adicional esta función también inicializa las variables que servirán de control para la actividad generada desde las páginas ZHTML.

Ahora pasamos a analizar los bloques funcionales que conforman este módulo sin ahondar en los detalles propios de la sintaxis introducida por RabbitWeb, ya que, como se ha mencionado, existe bibliografía al respecto.

```
//User groups declaration
#web_groups ADMIN_GROUP
#web_groups OPER_GROUP
#web_groups ALL_GROUP
```

Declaración de los grupos de usuarios que podrán acceder a las páginas Web de la interfaz

```
/******
Web variables shared with
main program variables
*****/
```

```
extern ad_inputs[8];
#web ad_inputs
extern qd_reading;
#web qd_reading
extern IC1counter;
#web IC1counter
extern IC2counter;
#web IC2counter
```

Referencias a variables del programa principal para que puedan estar accesibles al servidor Web

```
#web NAME
#web SAMPLEINTERVAL
/******
Exclusive Web variables
*****/
```

```
//Time
char strDateTime[20];
#web strDateTime
//Sample interval
unsigned int SampleInterval;
#web SampleInterval
//user password
char strPass[5];
#web strPass
//system name
char strName[20];
#web strName
```


Variables creadas especialmente para la interfaz Web, algunas funcionan como un buffer intermedio para proteger las variables de operación del programa, hasta que sea seguro actualizar. Nótese la declaración estándar y a continuación la directiva `#web` de RabbitWeb que hace accesible la variable al servidor Web.

```
/******
for interface web control
*****/
```

```
unsigned char webreply;
#web webreply
unsigned char admincmd;
#web admincmd auth=basic, digest groups=ADMIN_GROUP(rw),OPER_GROUP(ro)
unsigned char opercmd;
#web opercmd auth=basic, digest groups=ADMIN_GROUP(rw),OPER_GROUP(rw)
char webrefresh;
```

Este grupo de variables es utilizado para controlar diferentes estados de operación que provienen de las páginas producto de la interacción con el usuario. Nótese que algunas de estas variables están protegidas, lo que significa que solo los usuarios pertenecientes a determinado grupo pueden acceder a ellas.



 <p>CONTINEA Microprocesamiento modular + Conectividad</p>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 9 de 17

```
#web webrefresh
char webmaincmd;
#web webmaincmd auth=basic, digest groups=ADMIN_GROUP(rw),OPER_GROUP(rw)
```

```
/******
```

```
Physics resources
```

```
*****/
```

```
#ximport "myProjects\CoAN-007\IWeb\index.zhtml"           index_html
#ximport "myProjects\CoAN-007\IWeb\setup.zhtml"          setup_html
#ximport "myProjects\CoAN-007\IWeb\style.css"            style_css
```

```
/******
```

```
Functions to control
```

```
web sending values
```

```
*****/
```

```
//to control NAME and TIME
```

```
void opercmd_change(void)
```

```
{
    switch (opercmd)
    {
        case 'n': cfgSaveName(strName);break;
        case 't': set_time(strDateTime);break;
        default:break;
    }
    webreply=0;
}
```

Comienza la definición de las funciones que actúan como nexos entre la interfaz Web y el programa principal.

En este bloque se cargan en memoria de programa las páginas Web que ya tenemos que tener diseñadas. Nótese que además de las páginas zhtml, se carga en memoria también la hoja de estilos .css

```
//to control SAMPLEINTERVAL and PASSWORDS
```

```
void params_change(void)
```

```
{
    switch (admincmd)
    {
        case 's': cfgSaveSampleInterval( SampleInterval ); break;
        case 'o': cfgSavePassword(strPass, 1);
                  sauth_setpassword( 1, strPass );
                  break;
        case 'a': cfgSavePassword(strPass, 0);
                  sauth_setpassword( 0, strPass );
                  break;
        default:break;
    }
    webreply=0;
}
```

Esta función es llamada cuando alguna de las variables `strName` o `strDateTime` (declaradas arriba) cambian como producto de la acción del usuario desde la página Web. El llamado oportuno de funciones como estás es controlado internamente por el servidor http, y el vínculo "variable-función" se define por directivas `#web_update` como se verá más abajo.

Igual que la función precedente, pero con relación al cambio de las variables correspondientes. En este caso, lo más relevante es el cambio del password y su notificación al servidor Web mediante la función `sauth_setpassword()`;

```
//to control Refresh process of main page
```

```
void page_refresh(void)
```

```
{
    get_time( strDateTime, &strDateTime[11]);
    strDateTime[10]=' ';
    webrefresh--;
}
```


En este caso, esta función es invocada cuando la variable `webrefresh` cambia, esto es forzado desde la función de auto-refresh que fue definida para la página principal y aprovechamos este llamado periódico para actualizar el string que contiene la hora del equipo, de forma de hacer esto bajo demanda.

```
//to control QD reset button from main page.
```

```
void cmd_change(void)
```

```
{
    if (webmaincmd=='Q')
        qd_reading=0;
    webmaincmd=0;
}
```

La última función que interactúa con un botón de comando que se creó en la página principal, y es para resetear el contador del detector de cuadratura.

 <p>CONTINEA Microprocesamiento modular + Conectividad</p>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 10 de 17

```

/*****
linking variables with each handler function
*****/
//Main page
#web_update webrefresh          page_refresh
#web_update webmaincmd         cmd_change
//config page
#web_update SampleInterval, strPass, admincmd  params_change
#web_update strName, strDateTime              opercmd_change

```

Aquí podemos ver las directivas #web\_update que vinculan las variables accedidas desde la interfaz Web con las funciones que fuimos describiendo y que sirven como nexo entre la páginas ZHTML y nuestra aplicación.

```

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


```

Por último las tablas de mapeo de los tipos MIME y las de recursos para el servidor Web

```

//server directory, functions and SSI variables
SSPEC_RESOURCETABLE_START
SSPEC_RESOURCE_P_XMEMFILE("/", index_html, "RabbitWebServer", ADMIN_GROUP|OPER_GROUP|ALL_GROUP,0,SERVER_HTTP,SERVER_AUTH_BASIC | SERVER_AUTH_DIGEST),
SSPEC_RESOURCE_P_XMEMFILE("/index.zhtml", index_html, "RabbitWebServer", ADMIN_GROUP|OPER_GROUP|ALL_GROUP,0,SERVER_HTTP,SERVER_AUTH_BASIC|SERVER_AUTH_DIGEST),
SSPEC_RESOURCE_P_XMEMFILE("/setup.zhtml", setup_html, "RabbitWebServer", ADMIN_GROUP|OPER_GROUP,0,SERVER_HTTP,SERVER_AUTH_BASIC | SERVER_AUTH_DIGEST),
SSPEC_RESOURCE_XMEMFILE("/style.css", style_css),
SSPEC_RESOURCETABLE_END

```

	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 11 de 17

## Módulo principal

En cuanto al módulo principal (único archivo .c del proyecto) mostramos aquí solo el esqueleto y parte de los comentarios que aluden a una diferenciación de los bloques del programa.

La primera etiqueta que aparece es la que indica que debajo de sí misma están presentes las líneas de configuración IP del sistema y la invocación de las libraries que implementan el servidor Web y TCP/IP, por supuesto el usuario deberá modificar las línea de definición correspondientes a esta sección de acuerdo a su red.

```

/*****
*      NETWORK CONFIGURATION      *
*****/

```

Le sigue a esta sección la invocación de los módulos que diseñamos para este proyecto y que acabamos de describir :

```

/*****
*  application modules for this project  *
*****/

```

A continuación, se encuentra el grupo de variables globales que almacenan lo valores de simulación para mostrar en las páginas Web de la interfaz que nos ocupa:

```

/*****
*  variables for samples simulations      *
*****/

```

Seguidamente están las funciones que establecen los valores para las variables arriba mencionadas:

```

/*****
*  functions for samples simulations      *
*****/

```

```
main()
```

```
{
```

// Dentro del bloque main vemos inicializaciones varias, que en general se auto-explican:

```

    cfgInit();
    iweb_init();
    sock_init_or_exit(1);
    http_init();
    tcp_reserveport(80);

```


//Inmediatamente comienza el loop principal y una serie de costates que cumplen la función de simplemente //generar valores de simulación para las variables expuestas a través de la Web.

```

while(1)
{
    http_handler();//Web server attention...

    /*****
    *  Simulate AD Samples      *
    *****/
    costate
    { }
    /*****
    *  Simulate Quadrature Detector      *
    *****/
    costate
    { }
    /*****
    *  Simulate Input Capture      *
    *****/
    costate
    { }
}
}

```

	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 12 de 17

## Los recursos Web del proyecto

Habiendo descripto convenientemente el módulo `iweb.lib` pasamos a ocuparnos de las páginas ZHTML y posteriormente del archivo `.CSS` que en realidad constituyen los objetos que queremos destacar mayormente en esta nota.

### Página principal (`index.zhtml`)

Sería imposible mostrar con claridad ciertos puntos importantes del código de la página ZHTML sin antes realizarle, digamos, una disección, con fines de poder mostrar lo que realmente importa. A continuación vemos el resultado de esto y pasamos a los comentarios:

```

<html>
<head>
  <link type="text/css" href="style.css" rel="stylesheet"/>
  <script language="javascript" type="text/javascript">
    function send()
    {
      document.frmMain.webrefresh.value++;
      document.frmMain.submit();
    }
    function setcmd(cmd)
    {
      <?z if (auth($webmaincmd, "rw")) { ?>
        document.frmCmd.webmaincmd.value = cmd;
        document.frmCmd.submit();
      <?z } ?>
      <?z if (!auth($webmaincmd, "rw")) { ?>
        alert("No tiene permiso para ejecutar este comando");3
      <?z } ?>
    }
    function go(toPage)
    {
      window.clearTimeout(timer);
      window.location = toPage ;
    }
  </script>
</head>
<body onload="timer=setTimeout('send()',3000)">4
  <form action="index.zhtml" method="post" id="frmMain" name="frmMain">
    <input type="hidden" name="webrefresh" value=" <?z print(@webrefresh) ?>" />
  </form>
  <form action="index.zhtml" method="post" id="frmCmd" name="frmCmd">
    <input type="hidden" name="webmaincmd" value=" <?z print(@webmaincmd) ?>" />
  </form>

```

Mediante esta línea estamos vinculando la página zhtml

Comienzo del bloque con código JavaScript

Utilizada para refresco periódico de la página. Notese cómo se realiza la acción submit del formulario "frmMain" y se modifica el valor de la variable webrefresh (recuerde que webrefresh fue declarada en `iweb.lib`)

Envía el comando de reset del contador "QD". Nótese la combinación con el script ZHTML: si el usuario tiene permisos 'rw' para webmaincmd el HTML incluirá el código que resaltamos en verde, de lo contrario, incluirá el resaltado en rojo, impidiendo que la presión del botón que invoca a esta función realice el envío, y provocando que el usuario reciba su merecida advertencia mediante la función nativa "alert()"

Esta función simplemente redirecciona a la página indicada como parámetro. Su razón de ser es apagar el timer antes de redireccionar.

Esta línea en Javascript ejecutada en el evento onload de la página define su refresco periódico. SetTimeout es una función Javascript nativa; send() es una referencia a nuestra función definida arriba, 3000 son los milisegundos del periodo del timer que llamará a send() y timer es un objeto que realiza lo que su nombre indica.

Este formulario realiza el auto-refresh. La variable webrefresh está implementada mediante un elemento *input hidden*, que toma el valor enviado por el módulo `iweb.lib` (recordar la función `void page_refresh(void)`). La función `send()`, tal como vimos le aplica el metodo `submit` a este formulario, a su vez, `send()` es invocada por el timer que creamos arriba en el evento `onload`.

Este formulario realiza el envío de la variable webmaincmd que está implementada mediante un elemento *input hidden*, y toma el valor enviado por el módulo `iweb.lib` (recordar la función `void cmd_change(void)`). La función `setcmd()`, tal como vimos le aplica el metodo `submit` a este formulario, a su vez, `setcmd()` es llamada al registrarse el evento `onclick` del botón de comando, según podrá verse más debaio de estas manera: `onclick="setcmd('O');"`

<sup>3</sup> Ver [http://www.w3schools.com/JS/js\\_popup.asp](http://www.w3schools.com/JS/js_popup.asp) para aprender algo más sobre funciones pop-up

<sup>4</sup> Ver [http://www.w3schools.com/js/js\\_timing.asp](http://www.w3schools.com/js/js_timing.asp) para aprender algo más sobre los timers.

```
<?z print(@config.name) ?>
<td class="menu" onclick="go('setup.zhtml');">
```

El llamado a la función de redireccionamiento de página lo realizamos acá, cuando se recibe el evento click en el menú para ir a la página de configuraciones.

```
<td class="data"><a><?z printf("%.2f",@ad_inputs[0]) ?></a></td>
<?z printf("%d",@qd_reading) ?></a>
<?z print(@IC1counter) ?></a>
<?z print(@IC2counter) ?></a>
```

Estas son simples impresiones de las variables que declaramos y/o definimos y/o referenciamos en el módulo `iweb.lib`

```
<input class="" style="font-weight:bold;" type="button" value="Resetear contador QD"
onclick="setcmd('Q');" />
```

El tag que define al botón de comando para resetear el contador. Lo importante acá es el llamado a la función `setcmd()` mediante el evento `onclick` del botón.

Por último, aprovechamos la siguiente línea, mediante la cual mostramos la fecha y la hora del equipo, para remarcar la forma en que concretamente aplicamos el estilo a los documentos HTML mediante hojas de estilo CSS.

```
< a class="foot"><?z print(@strDateTime) ?></a></td>
```

```
</body>
</html>
```

La asignación del valor `"foot"` a la propiedad `Class` del tag `<a>` ( `<a class="foot">` ) define que dicho tag HTML será de la clase `"foot"`, heredando automáticamente todas sus propiedades. Y la clase `"foot"` está definida dentro de la hoja de estilos `"style.css"` para el elemento `<a>` de esta manera, pudiendo definirse una clase `"foot"` diferente para elementos HTML diferentes, por ejemplo:

```
A.foot{
}
TD.foot{
}
TABLE.foot{
}
and so on...
```

Así, el archivo CSS es sencillamente una colección de clases definidas tal cual se puede ver en estos ejemplos. Lo único que hay que saber es que propiedades y valores son válidos para cada elemento HTML.

```
A.foot
{
  FONT-WEIGHT: 300;
  COLOR: #000000;
  FONT-FAMILY: Tahoma, Verdana, sans-serif;
}
```

Sobre las hojas de estilo hay mucho para aprender, para esto pueden consultar muchos tutorial que hay disponibles en la web.<sup>5</sup>


## Página de configuraciones (setup.zhtml)

Es el turno ahora de examinar la página de configuraciones, que por su parte presenta algunos detalles interesantes que la página `index.zhtml` no posee. Con fines de no extendernos demasiado vamos a concentrarnos mayormente en las particularidades que esta página incluya, obviando las líneas que por su similitud con las explicadas en el apartado anterior no justifiquen su explicación:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="style.css" />
```

El enlace con nuestra hoja de estilos, tal como se ha hecho en `index.zhtml`

<sup>5</sup> Un tutorial sobre hojas de estilo puede encontrarse en <http://es.html.net/tutorials/css/>

	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 14 de 17

```

<script language="javascript" type="text/javascript">
  // add a zero in front of numbers<10
  function checkTime(i){
    return (i<10)? "0"+i: i;
  }
  function startTime()
  {
    var today=new Date();

    var d= checkTime( today.getDate() );
    var mo=checkTime( today.getMonth() + 1 );
    var y= today.getFullYear();

    var h= checkTime( today.getHours() );
    var mi= checkTime( today.getMinutes() );
    var s= checkTime( today.getSeconds() );

    var r=document.getElementById('strDateTime');6 7
    r.value= d+"/"+mo+"/"+y+" "+h+": "+mi+": "+s;
    var t=setTimeout('startTime()',500);

  }
  function send(key)
  {
    switch(key)
    {
      case 's':case 'x':
        document.frmAdmin.admincmd.value = key;
        if(document.frmAdmin.SampleInterval.value > 0 )
          document.frmAdmin.submit();
        else
          alert("Error: valor de tiempo negativo");
        break;
      case 'o': case 'a':
        var rol=(key=='o')?"Operador":"Administrador";
        document.frmAdmin.admincmd.value = key;
        if(document.frmAdmin.pass1.value == document.frmAdmin.pass2.value )
        {
          if ( confirm("Confirma modificar la clave de " + rol + " ?")8 )
            document.frmAdmin.submit();
        }
        else
          alert("Error: " + "\n" + "La repetición de la clave no es correcta");
        break;
      case 't':
      case 'n':
        document.frmOper.opercmd.value = key;
        document.frmOper.submit();
        break;
    }
    return;
  }
</script>
</head>
<body onload =" startTime();" >
<?z print(@config.name) ?>

```

Tag que da comienzo al bloque donde vamos a ubicar nuestro código JavaScript

Función auxiliar para mostrar en dos dígitos todos los valores de fecha y hora.

Esta función es la implementación de un reloj en pantalla que obtiene la fecha y hora del sistema.

La fecha y hora obtenida se introduce en una caja de texto que tiene id = "strDateTime". Nótese como se utiliza el método "getElementById()" del objeto "document" para obtener una referencia a la caja de texto que nos interesa y así poder escribir en ella.

Esta función culmina con la creación de un timer con período de 500ms que provoca una especie de recursividad, ya que fuerza el llamado permanente a esta función, manteniendo la hora actualizada segundo a segundo en la pantalla del browser, por supuesto, sin refrescos de página.


Esta es una función de validación básica, que es invocada por los botones de comando de los formularios de esta página. Mediante el switch, determina quién la llamó, para realizar a continuación la validación pertinente. Nótese el llamado a la función Javascript nativa "confirm()"

Muy similar al caso anterior. Pero en esta oportunidad, en vez de forzar el refresco de la página hacemos que al dispararse el evento "onload" se invoque la función startTime(); que como se ha visto en su última línea instancia un timer con un intervalo de 500ms que al expirar vuelve a llamar a esta misma función, que estaría actuando como una función reentrante.

<sup>6</sup> Vea el método **getElementById** en [http://www.w3schools.com/html/htmldom/met\\_doc\\_getelementbyid.asp](http://www.w3schools.com/html/htmldom/met_doc_getelementbyid.asp)

<sup>7</sup> Conozca un poco mas del objeto **document** en [http://www.w3schools.com/html/htmldom/dom\\_obj\\_document.asp](http://www.w3schools.com/html/htmldom/dom_obj_document.asp)

<sup>8</sup> Vea en [http://www.w3schools.com/JS/js\\_popup.asp](http://www.w3schools.com/JS/js_popup.asp) algo más sobre la función **confirm()** y otras similares.

 <p>CONTINEA Microprocesamiento modular + Conectividad</p>	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 15 de 17

```

<?z if(updating()) { ?>
  <?z if(!error()) { ?>
    <a class="ok"><?z if(@webreply==0) { ?>
      Comando ejecutado con &eacute;xito
    <?z } ?>
    <?z if(@webreply == -1) { ?>
      Errores en la ejecuci3n del comando
    <?z } ?>
  </a><br>
  <?z } ?>
  <?z if(error()) { ?>
    <a class="error">Errores en la ejecuci3n del comando
    <font color="#ff0000"><?z print(error($admincmd)) ?></font>
  </a><BR>
  <?z } ?>
<?z } ?>

```

El bloque delimitado por las líneas en negrita, está conformado por scripts ZHTML, que comprueban la existencia de errores para mostrar el mensaje según el caso.

Formulario que transporta la variables correspondientes al nombre del equipo y a la fecha y hora, el evento onclick de los botones realiza el llamado a la función `send()` pasndo como argumento un carácter a módo de código para que la función `send` identifique a quien realizó el llamado. A su vez este código es asignado por la función `send()` al elemento `input hidden` llamado `opercmd` que representa la variable con el mismo nombre en el módulo `iweb.lib` con la finalidad de que allí también se pueda conocer el origen del llamado.

```

<form id="frmOper" name="frmOper" method="post" action="setup.zhtml">
  <input type="hidden" name="opercmd" value=" <?z print(@opercmd) ?>" >
  <input style="text-align: left" name="strName" maxlength="20" value="<?z print(@config.name) ?>">
  <input class="boton" type="button" onclick="javascript:send('n');" >
  <input style="text-align: left" id="strDateTime" name="strDateTime" maxlength="19" value="dd/mm/aaaa
  hh:mm:ss" >
  <input class="boton" type="button" onclick="javascript:send('t');" >
</form>

```


Este formulario funciona con exactamente la misma filosofía que el anterior, solo que cambian obviamente las variables, pero la particularidad está en el script ZHTML resaltado en verde, ya que gracias a este logramos que el formulario no aparezca si el usuario que accedió a la página no pertenece a un grupo que posea permisos `rw` sobre la variable `admincmd`, de esta forma logramos fácilmente un eficaz control de acceso a los diferentes recursos de la páginas.

```

<?z if (auth($admincmd, "rw")) { ?>
  <form id="frmAdmin" name="frmAdmin" method="post" action="setup.zhtml">
    <input type="hidden" name="admincmd" value="<?z print(@admincmd) ?>" >
    <input style="text-align: left" type="password" name="strPass" id="pass1" size="4" maxlength= "4"
    value="*****">
    <input class="boton" type="button" onclick="javascript:send('a');" >
    <input class="boton" type="button" onclick="javascript:send('o');" >
    <input style="text-align: left" type="password" name="strPass" id="pass2" size="4" maxlength= "4"
    value="*****">
    <input name="SampleInterval" size="5" maxlength="4" value="<?z print(@config.sample_interval)
    ?>" >
    <input class="boton" type="button" onclick="send('s');" >
  </form>
<?z } ?>

</body>
</html>

```

	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
		CoAN-007
	<b>RabbitWeb y JavaScript, y también CSS</b>	Publicado: 00/00/0000
		Página 16 de 17

## Hoja de estilos (style.css)

Con respecto a la hoja de estilos, puede aclararse que esta es simplemente un recurso más que reside en el servidor Web, en este caso nuestro módulo Rabbit, y se puede cargar en la memoria flash del módulo con un directiva `#ximport`, exactamente de la misma forma que una página ZHTML o HTML común.

Ya hemos visto la forma en que podemos vincular la hoja de estilos con nuestras páginas, hay otras, pero por ahora con esta nos alcanza:

```
<head>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
```

y la forma de aplicar el estilo a los diferentes elementos de la página HTML:

```
<tr>
  <th class="header"></th>
</tr>
<tr>
  <td class="data"> </td>
</tr>
```

En cuanto al formato general dentro de las hojas de estilo, es como se puede ver en la imagen gentileza de <http://es.html.net/tutorials/css/lesson2.asp>

```
selector {property: value;}
```

↑  
A qué etiqueta(s) HTML se aplica la propiedad (por ejemplo, "body")

↑  
La propiedad, por ejemplo, podría ser el color de fondo ("background-color")

↙  
el valor de la propiedad "background-color" podría ser, por ejemplo, rojo ("#F0000", valor en hexadecimal)

Así en nuestro archivo CSS podemos tener definidas clases como estas:


<pre>td{   propiedad_A: valor_1; }</pre>	<p>Aplica a todo elemento <code>&lt;td&gt;</code> que ande dando vueltas por ahí sin que se le haya asignado estilo individualmente, es decir, sería como el estilo predeterminado para el <code>&lt;td&gt;</code></p>
<pre>td.data{   propiedad_A: valor_1; }</pre>	<p>Es una clase llamada "data" que es aplicable específicamente a elementos <code>&lt;td&gt;</code> y de aplicarse sobrescribiría al estilo por defecto.</p>
<pre>.data{   propiedad_A: valor_1; }</pre>	<p>Esta es una clase llamada "data" que es aplicable a cualquier elemento, siempre que las propiedades concuerden con las del elemento receptor del estilo.</p>

o, las llamadas pseudo-clases como estas:

```
a:link{      a:visited{      a:active{      a:hover{
}            }            }            }
```

Y eso es todo lo que tenemos para decir por el momento con respecto a los archivos .CSS



	<b>RCM4300 + RabbitWeb</b>	Nota de Aplicación
	<b>RabbitWeb y JavaScript, y también CSS</b>	CoAN-007
		Publicado: 00/00/0000
		Página 17 de 17

## Referencias

Las siguientes referencias corresponden a diferentes sitios que fueron consultados durante la realización de la nota y deberían consultarse para profundizar conocimientos.

### JavaScript

<http://www.w3schools.com/JS/default.asp>

### Hojas de estilo

<http://es.html.net/tutorials/css/>

### Rabbit TCP/IP Manual Vol.2

<http://www.rabbit.com/documentation/docs/manuals/TCPIP/UsersManualV2/index.html>

### RabbitWeb

<http://www.rabbit.com/documentation/docs/manuals/TCPIP/UsersManualV2/rabbitweb.html#1002674>

### RabbitWeb (Tutorial en castellano)

[http://www.cika.com.ar/soporte/Tutorials/CTU-006\\_RabbitWeb.pdf](http://www.cika.com.ar/soporte/Tutorials/CTU-006_RabbitWeb.pdf)