1 SEANCE WEB : Mise en œuvre de fonction CGI pour générer des pages web dynamiques avec le processeur Beck

Rappels des objectifs :

L'idée est de pouvoir générer dynamiquement une page en code HTML composée d' éléments fixes (dans le « squelette ») et d éléments variables (les données météorologiques qui évoluent en temps réel) Ainsi, le client WEB obtiendra dans son navigateur Internet (Internet Explorer) une page HTML avec des éléments dynamiquement modifiés par le contexte des paramètres météo.

2 Explication et topologie du mécanisme CGI du Beck

Le mécanisme normal mis en place pour servir une page en code HTML à un client, doté d'un navigateur WEB, est le suivant :

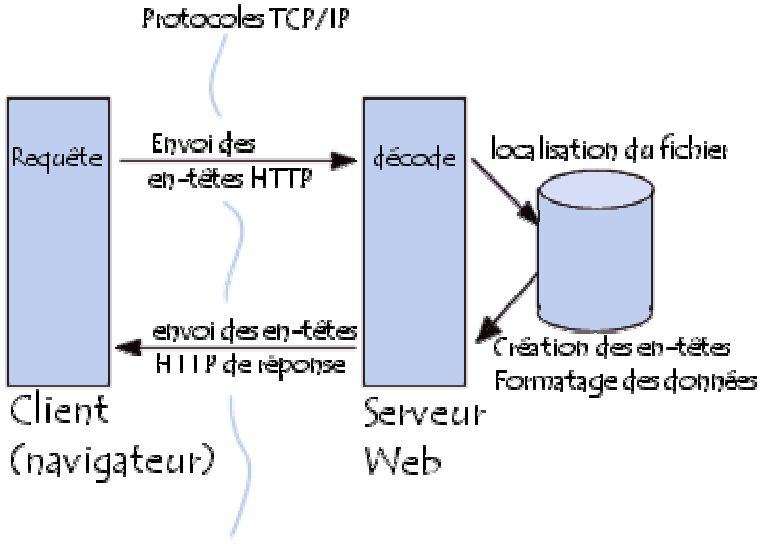
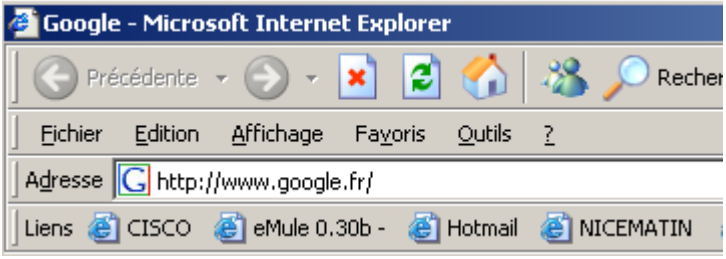


Figure 2-1 : mécanisme classique d'accès à une page HTML sur un serveur WEB

La séquence de communication classique entre le client et le serveur WEB est la suivante :

1) Le client émet une requête http depuis son navigateur WEB .Pour cela il indique, dans la barre d'adresse de son navigateur internet (IE, NETSCAPE..) ; l'URL de la page demandée.



2) La requête transite sur internet (Port de destination TCP 80) et arrive jusqu'au serveur WEB distant.

3) Le serveur WEB établit une connexion TCP avec ce client et analyse la requête contenu dans l'URL. Si cette requête porte sur une page WEB particulière (exemple : index.htm) le serveur WEB cherche cette page sur son disque interne et la récupère.

4) La page est alors renvoyée vers le client distant par la liaison TCP établie.

5) Une fois la page réceptionnée en totalité par le client (avec éventuellement des fichiers binaires d'image, de son de vidéo) la connexion TCP est coupée à l'initiative du serveur WEB

Dans le cas d'une page à générer dynamiquement en fonction du contexte du serveur.
Cette fois la page à renvoyer au client n'existe pas sur le système de fichier du serveur ou alors seul son squelette « vide » y est présent .
On va donc la générer, par une fonction écrite en C (un Callback), le code HTML à renvoyer au client distant :

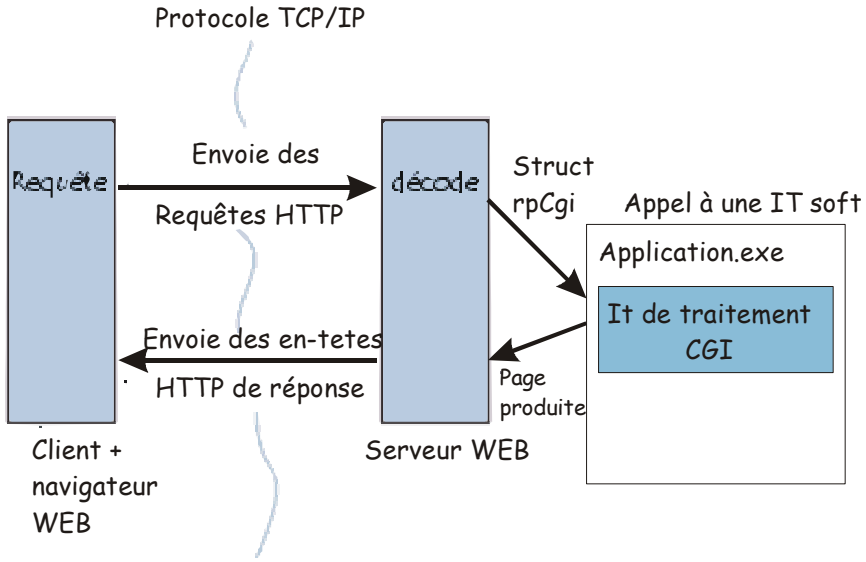


Figure 2-2 : mécanisme CGI utilisé pour la génération d'une page WEB par le Beck.

Séquence des opération du mécanisme CGI dans le Beck
1) le client envoie toujours une requête au serveur WEB, mais cette fois la requête porte sur une page html qui n'existe pas dans le disque dur du serveur.

2) Le serveur reçoit cette requête mais voit quelle porte sur un nom de page ne figurant pas dans son système de fichier.

3) Le serveur regarde alors si on lui a indiqué précédemment, d'appeler une fonction particulière (la fonction CGI) pour traiter cette demande particulière.

4) Si c 'est le cas, le serveur WEB transmet la demande à cette fonction d'interruption (IT de traitement CGI) et lui passant l'adresse d'une structure (rpCGI) contenant plein d'informations sur la demande du client :. @IP du client, URL demandée, éléments de la requête (dans le cas d'un formulaire), mot de passe ….

5) La fonction d'IT CGI récupère cette structure et génère une page HTML (tableau de char) correspondant à la demande. On pourra en profiter pour y inclure des éléments dynamiques comme la température, la pression atmosphérique…..

6) La fonction d'IT CGI retourne l'adresse du buffer contenant la page HTML générée par la même structure rpCgi au web serveur. Elle indique également qu'il faut servir cette page au client.

7) Le serveur web génère enfin une réponse au client en utilisant le buffer passé par la fonction d'IT CGI.

2.1 Exemple de code permettant la mise en place de ce mécanisme :

2.1.1 Création d'un squelette de page html

1) On a créé un squelette de page HTML qui est sauvegardé sur le disque A : du Beck :
Cette page se nomme « pagecgi.htm »

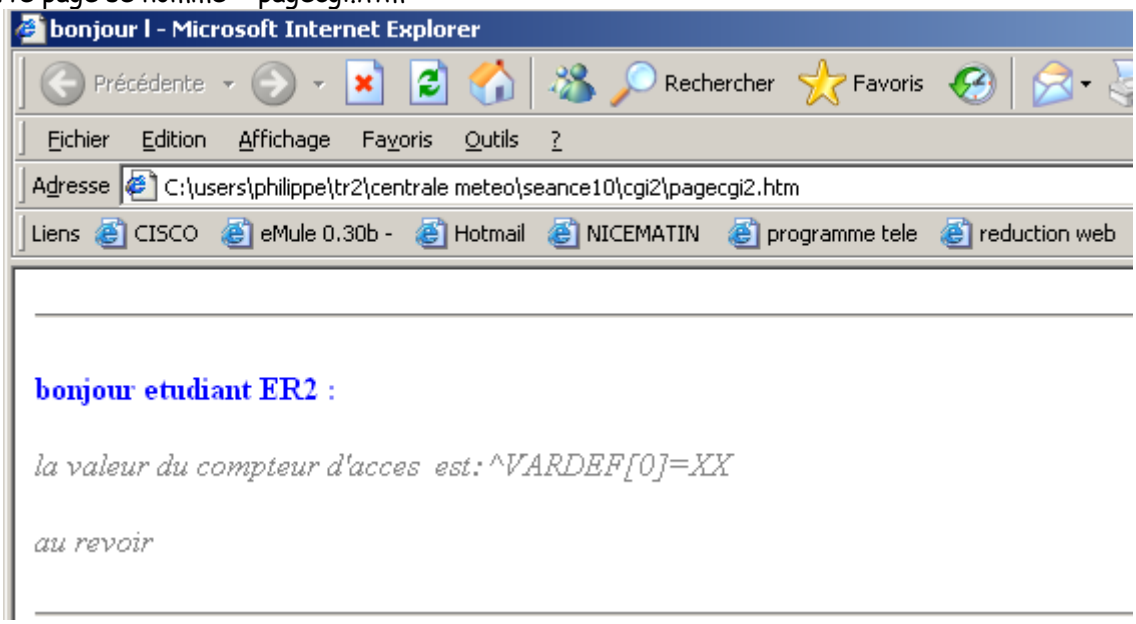


Figure 2-3 : squelette de page HTML à servir au client WEB

Cette page est sauvegardée sur le disque dur A:\ du processeur Beck. On y a introduit un élément de formatage de repérage (une balise) «^VARDEF[0]=XX » qui sera plus tard remplacée par une valeur dynamique. On a chargé cette page en mémoire dans un tableau de char 'ma_chaine3' lors de l'initialisation de l'application

2.1.2 Installation de cette fonction CGI auprès du serveur WEB principal.

Pour que ce mécanisme puisse fonctionner, il faut tout d abord enregistrer (installer) cette fonction CGI auprès du serveur. Ceci se fait dans l'application principale suivante:
Ouvrir le projet borland CGI1.ide

```
void main( void) {
  CGI_Entry My_cgiEntry;          // Holds information on the CGI function
  BIOS_Set_Focus(FOCUS_APPLICATION);

   printf( "\r\nCountCGI - ER2 example" "\r\n" );

    ma_chaine3 = Gen_HtmlCode_From_File("a:\\pagecgi2.htm",tab_balise,1);

  // ************ installation de la fonction CGI ******************
  My_cgiEntry.PathPtr = "compteur";  //Name of the page requested by client
  My_cgiEntry.method = CgiHttpGet;       // HTTP method
  My_cgiEntry.CgiFuncPtr = my_cgiFunction;//Function called:browser request
  CGI_Install( &My_cgiEntry );// installation fonction CGI

   // ********* attente de fin d'application ******************
    while (BIOS_kbhit()==0)    {
          RTX_Sleep_Time( 100 );
   }
   // ******************  Remove CGI function  ***************
   CGI_Delete( My_cgiEntry.PathPtr ); // remove CGI entry
   printf( "\r\nGood bye!" );
   BIOS_Set_Focus(FOCUS_SHELL);
  }
```

Dans ce code se trouve plusieurs parties indépendantes :

1) Chargement du squelette depuis le disque A:\ vers un tableau de char ma_chaine3

```
ma_chaine3 = Gen_HtmlCode_From_File("a:\\pagecgi.htm",tab_balise,1);
```

On fait appel à une fonction d'une bibliothèque spécifique HTML.C / HTML.H à laquelle on indique le chemin d'accès à la page html (ici la page s'appelle « pagecgi2.htm » et on lui passe un tableau de structures (tab_balise[]) et le nombre de balise à rechercher dans la page html.
Ce tableau de structure est basé sur le modèle suivant :

```
char * ma_chaine3;
var_field_t tab_balise[1];  //une balise est présente dans le squelette
```

Dans ce tableau (tab_balise[]) chaque case est une structure de deux éléments repérant la l'adresse (⇔position) et la longueur de chaque balise dans ma_chaine3. Dans notre exemple on a placé une balise ^VARDEF[0]=XX c'est à dire que la balise n°0 possède 2 caractères à modifier (les 2 X)

2) On paramètre la fonction CGI à installer puis on l'installe auprès du Web Server.
On créé une structure My_cgiEntry pour enregistrer cette entrée CGI auprès de l'OS.

```
CGI_Entry My_cgiEntry;         // Holds information on the CGI function
My_cgiEntry.PathPtr = "compteur";  // Name of the page requested by client
My_cgiEntry.method = CgiHttpGet;       // HTTP method
My_cgiEntry.CgiFuncPtr = my_cgiFunction; // Function called on browser
request
   CGI_Install( &My_cgiEntry );// installation fonction CGI
```

Dans cet exemple la page qui sera réclamée par le client Web se nomme « compteur ». Cette requête est générée par le client au travers de son navigateur :

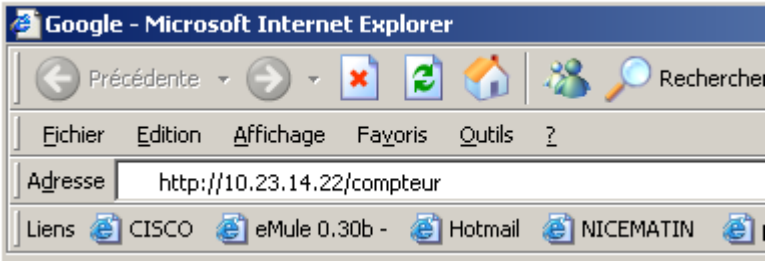


Figure 2-4 : exemple de requête envoyée par le navigateur du client pour activer notre CGI

On indique, dans la structure My_cgiEntry, l'**URL** de cette requête (qui doit provoquer l'activation de notre CGI), la méthode employée pour traiter cette requête (ici ce sera toujours CgiHttpGet) et le nom de la fonction à appeler si une telle demande apparaît (ici notre fonction CGI se nomme my_cgiFunction).
On installe cette nouvelle « entrée CGI » auprès du serveur WEB par la fonction CGI_INSTALL() en lui passant l'adresse de la structure que l'on vient de remplir.

3) Puis la fonction principale se met en attente infinie ou tant qu'une touche n est pas saisie dans la console Telnet

```
// ********* attente de fin d'application ******************
    while (BIOS_kbhit()==0)    {
          RTX_Sleep_Time( 100 );
   }
```

4) On veut arrêter l'application › Il faut donc désinstaller la fonction CGI du serveur WEB.

```
// ******************  Remove CGI function  ***************
   CGI_Delete( My_cgiEntry.PathPtr );
```

On fait appel à une fonction de l'API qui permet cette désinstallation. Il suffit de lui indiquer le nom de l'entrée CGI à supprimer. Ce nom avait été stocké dans la structure cgiEntry.PathPtr lors de son installation.
A partir de maintenant les requêtes du client portant sur la page « compteur » resteront sans réponse car le web server ne connaît plus cette entrée.

2.1.3La fonction d'IT CGI est construite sur le modèle suivant :

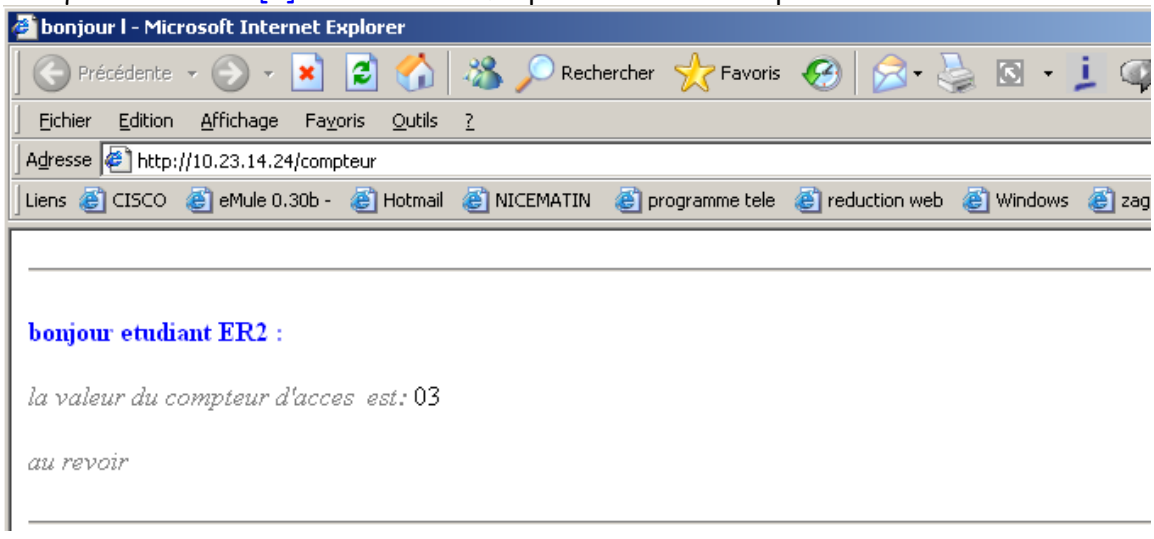
```
void huge _pascal my_cgiFunction( rpCgiPtr CgiRequest )
{
      //*****************************************************************
   //patch ma_chaine3 with counter value and create ma_chaine4
   //Fill CgiRequest structure with answer for the web server
   //*****************************************************************
  char ma_chaine4[20];
  sprintf (ma_chaine4,"%d",count);// convert count as ascii string
  Html_Patch (tab_balise,0,ma_chaine4);// patch first label with dyn.string
  count++;

  CgiRequest->fHttpResponse = CgiHttpOk;
  CgiRequest->fResponseBufferPtr = ma_chaine3;
  CgiRequest->fResponseBufferLength = strlen( ma_chaine3);
}
```

Dans cette fonction , le buffer `ma_chaine3` contient le squelette d'origine de la page html chargé depuis le disque. Dans la fonction on a un élément dynamique (`count`) qui doit être inséré dans le squelette à la place de l'élément de la balise n°0 (^VARDEF[0]=XXXXX) .
On convertit 'count' par la fonction `sprintf()` (analogue à un `printf()` mais dirigée vers une string plutôt que vers la console) :Le résultat de ce `sprintf()` donne naissance à `ma_chaine4`.

On utilise une fonction **`HTML_Patch()`** de la bibliothèque (html.c/html.h) pour indiquer que l'on veut remplacer la balise n°0 par notre nouvelle chaine (`ma_chaine4`) à l'intérieur du squelette de la page présente dans `ma_chaine3`.
Lorsque la requête du client arrive au serveur web, ce dernier active la fonction `my_cgiFunction()` et lui passe l'adresse de la structure CgiRequest (passage par adresse).
On indique enfin au serveur WEB du Beck de servir ce nouveau tableau de char (ma_chaine3) au client :

```
   CgiRequest->fHttpResponse = CgiHttpOk;
   CgiRequest->fResponseBufferPtr = ma_chaine3;
   CgiRequest->fResponseBufferLength = strlen( ma_chaine3);
```

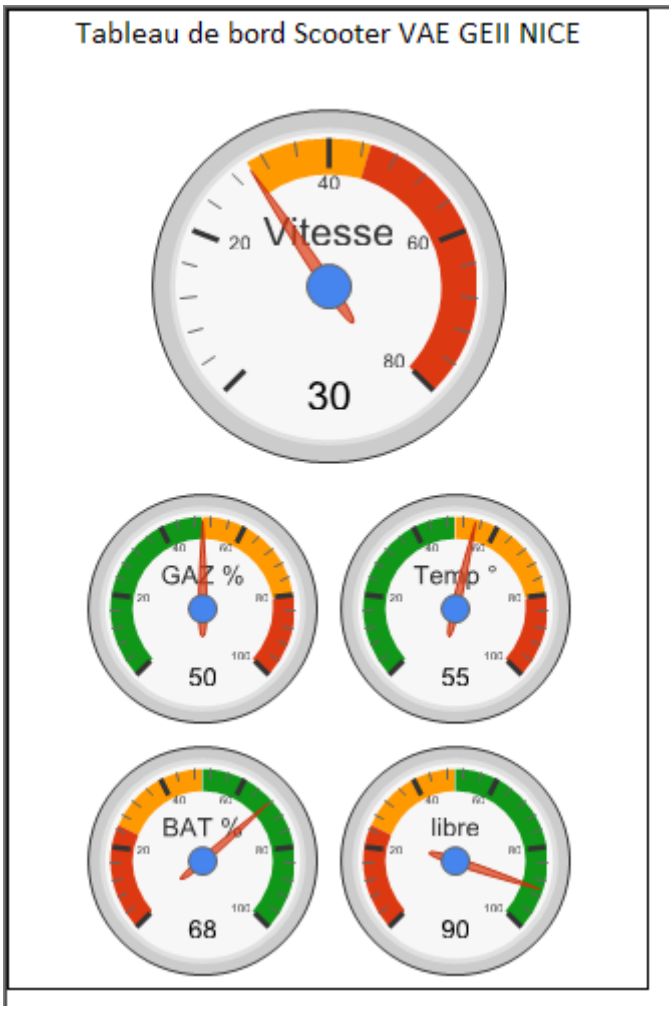
La fonction CGI se termine en rendant la main au serveur WEB et le client reçoit sa page WEB dans laquelle ^VARDEF[0]=XX a été modifié par la valeur du compteur count.



2.2 Utilisation de Google Graph pour afficher des compteurs

Google mets à disposition une bibliothèque de fonction graphique en ligne qui permet d'inclure dans son site web la possibilité de générer des graphiques dynamiquement à partir de variables. La documentation est disponible sous ce lien : http://code.google.com/apis/chart/

Le graph n'est pas fabriqué en local mais directement sur le serveur Google auquel on passe un paramètre qui est la valeur associée à chaque quadrant. Il faut donc une liaison internet externe pour pouvoir visualiser le graph dans tous les cas.
Parmi les graphs possibles, un est intéressant puisqu' il permet de dessiner des graphs de type jauge (compteurs)



On fournit un modèle de page html (jauge2.htm) dans lequel on a inséré le dessin de 3 graphs différents (3 fonctions drawChart1(),drawChart2() et drawChart3().

Un graph représente une ligne affichée dans la page html. Pour chaque graph on peut définir la taille, les labels ainsi que des zones colorées au travers d'options
Exemple :

```
var options =
{majorTicks:[0,20,40,60,80,100],width: 300,
height: 130, greenFrom: 0,
greenTo:50,redFrom: 80, redTo: 100,
           yellowFrom:50, yellowTo: 80,
minorTicks: 5,max:100};
```

Ces options permettent ainsi de définir les graduations, la largeur et la hauteur des quadrants, les limites des zones colorées ainsi que le maximum du graphique.
Plus de détails sur ces options ici
http://code.google.com/apis/chart/interactive/docs/gallery/gauge.html#Configuration_Options

enfin pour définir une valeur associée à un quadrant on utilise le code suivant :

```
function drawChart1() {
 var data = new google.visualization.DataTable();
 data.addColumn('string', 'Label');
 data.addColumn('number', 'Value');
 data.addRows(1);
 data.setValue(0, 0, 'Vitesse');
 data.setValue(0, 1, 50);
 //data.setValue(0, 1, ^VARDEF[0]=XX );
```

Dans cet exemple on a défini 1 seul quadrant (data.addRows(1)) avec une valeur associée =50. Si maintenant on décommente la dernière ligne, on pourra utiliser le Beck pour patcher cette valeur avec un élément dynamique du code (c est le rôle de la balise ^VARDEF[0]=XX)

Code complet de cette application : compteur.c

```
/*****************************************************************************
* Includes
*****************************************************************************/
#include <stdio.h>
#include "html.h"
#include <string.h>
#include <clib.h>

/*****************************************************************************
* Global variables
*****************************************************************************/
unsigned long count;              // Counter variable that will be displayed by CGI
char * ma_chaine3;
var_field_t tab_balise[2];  //une balise est présente dans le squelette


/*****************************************************************************
* cgiFunction()
* This function will be executed when a browser requests the page named "countcgi".
* It constructs a small HTML page containing the actual value of the counter
* variable and returns it to the webserver.
*****************************************************************************/
void huge _pascal my_cgiFunction( rpCgiPtr CgiRequest )
{  //*****************************************************************
   //patch ma_chaine3 with counter value and create ma_chaine4
   //Fill CgiRequest structure with answer for the web server
   //*****************************************************************
  char ma_chaine4[20];
   sprintf (ma_chaine4,"%d",count);// convert count as ascii string
   Html_Patch (tab_balise,0,ma_chaine4); // patch first label with dyn.string
   count++;

   CgiRequest->fHttpResponse = CgiHttpOk;
   CgiRequest->fResponseBufferPtr = ma_chaine3;
   CgiRequest->fResponseBufferLength = strlen( ma_chaine3);
}


/*****************************************************************************
* main()
*****************************************************************************/
unsigned long count;          // Counter variable that will be displayed by CGI
char * ma_chaine3;

void main( void)
{
  CGI_Entry My_cgiEntry;         // Holds information on the CGI function
  BIOS_Set_Focus(FOCUS_APPLICATION);
  printf( "\r\nCountCGI - ER2 example" "\r\n" );

   //*****************************************************
   //load html page from disk  and localize position of target to patch
   //*****************************************************
    ma_chaine3 = Gen_HtmlCode_From_File("a:\\pagecgi2.htm",tab_balise,1);

  // ************ installation de la fonction CGI ******************
   My_cgiEntry.PathPtr = "compteur";        // Name of the page requested by client
   My_cgiEntry.method = CgiHttpGet;       // HTTP method
   My_cgiEntry.CgiFuncPtr = my_cgiFunction;  // Function called on browser request
   CGI_Install( &My_cgiEntry );// installtion fonction CGI

   // ********* attente de fin d'application ******************
    while (BIOS_kbhit()==0)    {
          RTX_Sleep_Time( 100 );
   }

   // ******************  Remove CGI function  ***************
   CGI_Delete( My_cgiEntry.PathPtr );

   printf( "\r\nGood bye!" );
   BIOS_Set_Focus(FOCUS_BOTH);
  }
```