Utilisation des classes de PourCGI par Michel Michaud, version 2002-11-23

Les fichiers `PourCGI.h` et `PourCGI.cpp` rendent disponibles deux classes et une fonction libre qui permettent de faire facilement des applications CGI simples (les demandes doivent être pour des pages HTML). Une classe s'occupe de l'accès aux paramètres en unifiant les diverses possibilités de CGI. Elle permet en plus d'empêcher les accès concurrents aux données. L'autre classe permet de copier et compléter facilement des fichiers HTML pour les réponses aux requêtes. La fonction libre s'occupe de faire respecter le protocole HTTP pour la réponse.

Les classes sont prévues pour fonctionnement sous Windows avec le *Serveur Web personnel* (*Personal Web Server*, *PWS*) ou *Internet Information Server* (*IIS*). Elles n'ont pas été testées dans un autre environnement.

Classe `ClInfoCGI`

La classe `ClInfoCGI` permet d'accéder facilement aux paramètres reçus, indépendamment de la méthode d'envoi (*post* ou *get*) et du type d'encodage[1] utilisé. Les formulaires envoyés devraient contenir un champ caché (`<input type="hidden"...`) ayant un `name="Requete"`, avec un attribut `value` qui permettra de déterminer l'opération demandée par le formulaire spécifique (s'il y a plusieurs possibilités). Il ne faut déclarer qu'une seule variable de ce type (qu'on passe en paramètre par référence constante au besoin).

Fonctions membres principales

```
ClInfoCGI(const std::string& p_nomFicSemaphore="semaphore.txt");
```

Le constructeur s'occupe de trouver les paramètres et d'en normaliser l'accès. Au besoin, la fonction `DemarrerReponseHTTP` est appelée. Par défaut, mais on peut spécifier un autre nom de fichier, le fichier `"semaphore.txt"` est ouvert et gardé ouvert pendant l'exécution du programme afin d'empêcher des accès concurrents aux données; il est fermé mais n'est pas détruit à la fin de l'exécution.. (voir Gestion des erreurs)

```
std::string Requete();
```

Cette fonction renvoie la valeur du paramètre `"Requete"`.

```
std::string ValeurStringDe(const std::string& p_nomParam);
char ValeurCharDe(const std::string& p_nomParam);
int ValeurIntDe(const std::string& p_nomParam);
long ValeurLongDe(const std::string& p_nomParam);
double ValeurDoubleDe(const std::string& p_nomParam);
```

Ces fonctions renvoient la valeur du paramètre demandé dans un des types de base. Si on demande une valeur numérique d'un paramètre qui est simplement du texte, la valeur 0 est renvoyée. La valeur en `char` renvoie simplement le premier caractère ou `'\0'` si la valeur est vide.

```
bool EstUnChar(const std::string& p_nomParam);
bool EstUnInt(const std::string& p_nomParam);
bool EstUnLong(const std::string& p_nomParam);
bool EstUnDouble(const std::string& p_nomParam);
```

Ces fonctions renvoient `true` si la valeur du paramètre peut être convertie sans perte ni débordement dans le type demandé par les fonctions `ValeurTypeDe`.

Gestion des erreurs

Il y a trois types d'erreurs qui sont gérés par la classe `ClInfoCGI` : l'exécution du programme sans CGI, le *content-type* non supporté et l'échec de l'ouverture du fichier sémaphore. Dans les trois cas, le pseudo paramètre `Erreur` contiendra un message approprié, qu'on peut obtenir directement par la fonction spéciale `std::string Erreur()` ou par `ValeurStringDe("Erreur")`.

---

[1] Seuls les types de contenu standards sont supportés, `enctype="text/plain"` n'est **pas** supporté. Pour les requêtes *post*, on peut donner `enctype="multipart/formdata"` ou `enctype="application/x-www-form-urlencoded"`, ce dernier étant la valeur par défaut, moins efficace pour les formulaires contenant beaucoup de données. Les requêtes *get* sont toujours en *url encoded*. En général, on devrait employer des *post* en *formdata* pour les formulaires.

La fonction membre `bool EnCGI();` permet de vérifier que le programme a bien été exécuté dans un contexte CGI. Si ce n'est pas le cas, le pseudo-paramètre `Erreur` est le seul à contenir une valeur et il vaudra « Appel du programme sans CGI. ».

Si le type de contenu n'est pas accepté, le pseudo-paramètre `Erreur` est le seul à avoir une valeur et il vaudra « enctype inconnu (attention, text/plain non supporté). ».

Si l'ouverture du fichier sémaphore ne réussit pas, le constructeur ressayera jusqu'à 100 fois, en mettant un temps d'attente aléatoire entre les essais, ce qui totalisera entre 10 et 30 secondes. S'il ne réussit pas à ouvrir le fichier malgré tout, il lève une exception de type `ClInfoCGI::ClSemaphoreNonObtenu` et le paramètre `Erreur` vaudra « Système occupé, opération annulée pour prévenir les accès concurrents. ». Si le programme ne s'occupe pas de l'exception, le destructeur de la classe enverra automatiquement une page HTML simple indiquant que le système est occupé et qu'il faut réessayer plus tard. Par contre, si le programme traite l'exception et veut envoyer une page HTML personnelle en conséquence, il faudra appeler la fonction membre `Annuler()` pour éviter que le message automatique ne soit envoyé aussi.

Autres fonctions membres

```
int NbParametres();
```

Renvoie le nombre de paramètres, y compris `"Requete"` et `"Erreur"` le cas échéant.

```
std::string Nom(int p_noParam);
```

Renvoie le nom d'un paramètre selon son numéro. La liste est en ordre lexical et numérotée à partir de 0.

```
void AfficherListeDesValeurs();
```

Cette fonction est surtout utile pendant le débogage. Elle fait apparaître la liste de tous les paramètres, noms et valeurs, en ordre lexical sous la forme `"nom" = "valeur"`. Le texte est affiché dans un bloc `<samp></samp>`, un paramètre par ligne et le deux items (nom et valeur) sont encodées avec `ClCopieHTML::EncodeHTML` avant l'affichage (voir plus loin).

Classe `ClCopieHTML`

La classe `ClCopieHTML` permet de compléter et renvoyer des fichiers HTML en réponse aux requêtes CGI. Le principe consiste à faire un fichier HTML complet à l'exception des parties (*segment*) qui doivent être complétées selon les données courantes lors de la requête. On note ces segments du fichier sous la forme `[[[NomDuSegment]]]`. La classe permet de recopier facilement le fichier jusqu'à un tel segment et permet de savoir le nom des segments à compléter.

Fonctions membres de base

```
ClCopieHTML(const std::string& p_nomFichier);
```

Ce constructeur ouvre le fichier HTML indiqué; il peut contenir des segments à compléter sous la forme `[[[NomDuSegment]]]`. Si le fichier n’est pas accessible, un message HTML à cet effet est automatiquement renvoyé. Au besoin, la fonction `DemarrerReponseHTTP` est appelée.

```
bool Transcrire();
```

Cette fonction transcrit ou continue de transcrire le fichier HTML jusqu’au prochain segment à compléter. Elle renvoie `true` pour indiquer qu’il y a un segment à compléter, `false` si la transcription est terminée.

```
std::string NomSegment();
```

Cette fonction renvoie le nom du segment (sans les []) qui a fait arrêter la dernière transcription.

Autres fonctions membres

```
static std::string EncodeHTML(std::string p_texte);
```

Cette fonction peut être utilisée lors de l'écriture des données des segments s'il est possible que les données contiennent des codes HTML qu'on ne veut pas voir interprétés. La valeur de retour ne contient pas de & ni de < : ils sont remplacés par les codes `&` et `<` respectivement.

```
static std::string EncodeURL(std::string p_texte);
```

Cette fonction peut être utilisée lors de l'écriture des données des segments s'il faut composer un URL et qu'il est possible que les données contiennent des caractères invalides dans ceux-ci. La valeur de retour contient l'encodage url de ces caractères (un % suivi du code hexadécimal du caractère).

Fonction `DemarrerReponseHTTP`

```
void DemarrerReponseHTTP();
```

La fonction libre `DemarrerReponseHTTP` est disponible si on veut faire tout le traitement manuellement, elle s'occupe simplement d'envoyer la séquence initiale nécessaire pour la réponse HTTP. Il est inutile d'appeler manuellement cette fonction si on utilise `ClInfoCGI` ou `ClCopieHTML`.

EXEMPLES

Programme simple (sans `ClCopieHTML`) montrant les paramètres d'un formulaire

```
int main()
    {
    ClInfoCGI infoCGI;

    cout << "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n"
            "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\"\n"
            " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">\n"
            "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
            "<head><title>Informations sur le formulaire soumis</title>\n"
            "</head>\n"
            "<body>\n";

    if ( ! infoCGI.Erreur().empty())
        cout << "<h1>Erreur : " << infoCGI.Erreur() << "</h1>\n";

    if (infoCGI.NbParametres() != 0)
        {
        cout << "<h1>Liste de tous les paramètres reçus</h1>\n"
                "<p>\n";

        infoCGI.AfficherListeDesValeurs();

        cout << "</p>\n";
        }

    cout << "<hr />\n"
            "</body>\n"
            "</html>";
    }
```

Note : les `\n` qui sont écrits ne sont pas nécessaires, mais, au besoin, ils permettent de visualiser plus agréablement le code HTML produit.

Structure typique du programme principal pour CGI ayant plusieurs fonctions possibles

```
int main()
    {
    ClInfoCGI infoCGI;

    if (infoCGI.Requete() == "Ajout")
        AjouterMachin(infoCGI);
    else
    if (infoCGI.Requete() == "Liste")
        AfficherListeDesMachins(infoCGI);
    else
        AfficherPageErreur();
    }
```

Fonction transcrivant un fichier HTML complet

```
void AfficherPageErreur()
    {
    ClCopieHTML copieur("Erreur.html");
    copieur.Transcrire(); // On suppose qu'il n'y a pas de segment...
    }
```

Fonction transcrivant un fichier HTML à compléter

```
void AfficherListeDesMachins(const ClInfoCGI& p_infoCGI)
    {
    ClCopieHTML copieur("Liste.html");

    while (copieur.Transcrire())
        {
        string nomSegment= copieur.NomSegment();

        if (nomSegment == "DateDemandée")
            cout << p_infoCGI.ValeurStringDe("date");
        else
        if (nomSegment == "Liste")
            {
            if (NbDeMachins() == 0)
                {
                cout << "<tr><td colspan=\"2\">"
                     << "Aucun machin inscrit à la date demandée</td></tr>";
                }
            else
                {
                for (int i= 0; i != NbDeMachins(); ++i)
                    {
                    cout << "<tr><td>"
                         << ClCopieHTML::EncodeHTML(Machin(i).description)
                         << "</td><td>" << Machin(i).quantite << "</td></tr>";
                    }
                }
            }
        else
            assert(false); // Un segment non prévu !
        }
    }
```

Fichier HTML (Liste.html), utilisé dans la fonction précédente

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
<title>Liste des machins inscrits</title>
</head>

<body>
    <h1>Liste des machins inscrits</h1>
    <h2>en date du [[[DateDemandée]]]</h2>

    <table border="1">
        <tr><th>Description</th><th>Quantité en stock</th></tr>
        [[[Liste]]]
    </table>

    <p><a href="MonCGI.exe?Requete=Liste">Refaire la liste</a></p>
    <p><a href="default.html">Retour à la page d'accueil</a></p>
</body>
</html>
```

Programme principal avec gestion complète des erreurs d'accès concurrent

```
int main()
    {
    try
        {
        ClInfoCGI infoCGI;

        if (infoCGI.Requete() == "Ajout")
            AjouterMachin(infoCGI);
        else
        if (infoCGI.Requete() == "Liste")
            AfficherListeDesMachins(infoCGI);
        else
            AfficherPageErreur();
        }
    catch (ClInfoCGI::ClSemaphoreNonObtenu)
        {
        AfficherQuOnReessayeraPlusTard();
        infoCGI.Annuler();
        }
   }
```

Utilisation avec VC 6

Il est conseillé de copier localement les fichiers `PourCGI.h` et `PourCGI.cpp`, puis d'inclure le fichier d'en-tête par `#include "PourCGI.h"`. On devrait donc l'inclure après tous les fichiers standards (entre < >). De plus, il faut ajouter `PourCGI.cpp` au projet (l'ouvrir et demander une compilation, VC propose l'ajout).