Serveur http

Documentation technique de l'API

BARBOT Julien	barbot_u
COTTEREAU Gaëtan	cotter_g
PAUMARD Cédric	paumar_c
POUILLET Guillaume	pouill_g
THOMAS David	thomas_d

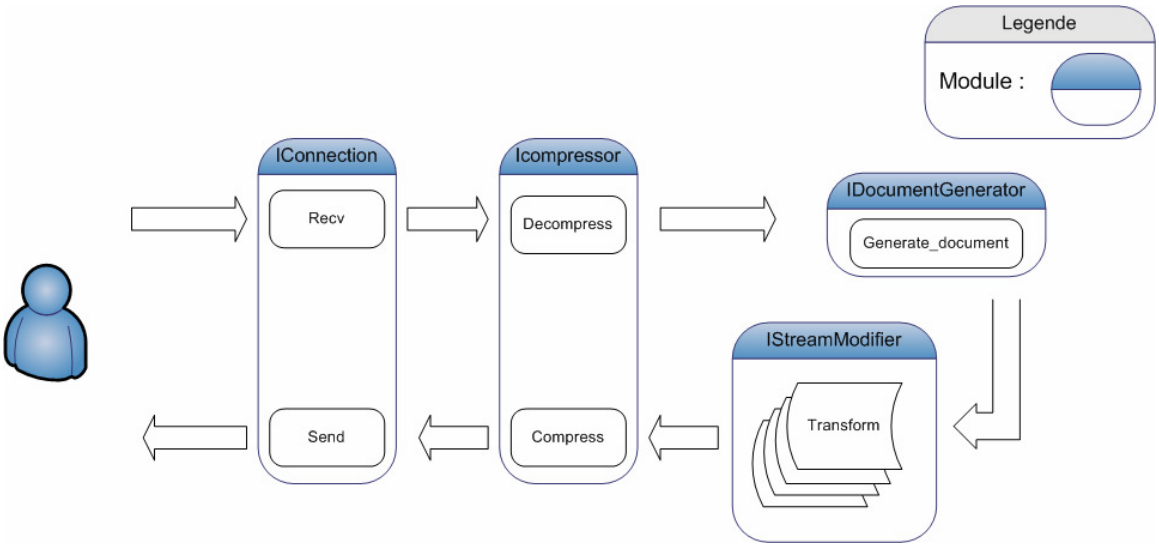
15/12/2005

1 Table des matières

2 Fonctionnement général

Le ZIIS a une architecture monolithique c'est-à-dire que toutes les fonctions de base (gestion des sockets, des threads, parse du HTTP, …) sont regroupées dans le programme. Les modules ne peuvent être ajoutés qu'à des endroits précis de la chaine de traitement d'une requête, à savoir :

- Lecture / écriture sur le réseau
- Compression / décompression de données
- Génération de la réponse



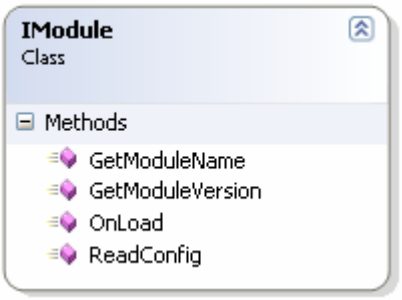
3 Le module

C'est une bibliothèque partagée (.dll sous windows, .so sous unix). Il doit contenir une classe héritant, dans un premier temps d'IModule, puis d'une des autres interfaces décrites ci-dessous (IConnection, ICompressor, IGenerator et IStreamModifier). Enfin, il nécessite la présence d'une fonction nommée « GetNewInstance » permettant l'instanciation de cette classe.

4 Les interfaces

Le rôle d'un module est définit par l'interface dont il hérite. Les interfaces disponibles sont les suivantes :

4.1. Interface IModule



Cette interface définit les bases d'un module et permet de le configurer. Tout module devra **impérativement** hériter d'IModule pour être considéré comme valide.

Attention un module n'est instancié qu'une seule fois par serveur. Toutes les threads de ce serveur utiliseront cette instance. Elles ne doivent donc pas utiliser des variables de classes contenant des données sur un client à moins de protéger les données d'accès simultanés.

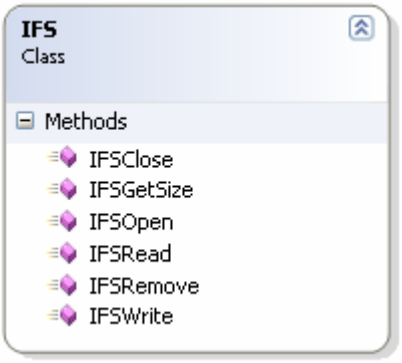
OnLoad		
Permet d'initialiser le module. Cette méthode est appelée au chargement du module afin de l'initialiser. Elle fourni au module l'interface IFS lui permettant d'utiliser le système de fichier simplifié du zia. Cette méthode n'est appelée qu'une seule fois pour finaliser le chargement du module. Cette méthode est la première à être appelée.		
Argument		
IFS *	Ifs	Pointeur sur l'interface IFS (gestion des fichiers).

ReadConfig		
Permet de (re-)charger la configuration du module. Cette fonction doit pouvoir être exécutée à tout moment. Renvoie false si le module n'a pas accès aux ressources nécessaires pour fonctionner correctement. Dans ce cas, le serveur n'utilisera pas ce module.		
Argument		
const char *	File	Chemin vers le fichier contenant la configuration du module
Retour		
bool	Indique si le (re-)chargement de la configuration s'est bien effectué.	

GetModuleName	
Fournit le nom du module. Il est utilisé, avec la version, pour l'interface homme/machine de gestion des modules.	
Retour	
const char *	Chaîne contenant le nom du module

GetModuleVersion	
Renvoie la version du module	
Retour	
const char *	Chaîne contenant la version du module

4.1.1 Interface IFS



Cette interface, implémentée par le serveur, fourni aux modules une gestion simplifiée des fichiers.

IFSOpen		
Ouvre le fichier File avec le droit en lecture ou en append suivant le flag spécifié. Cette méthode retourne un ID sur le fichier.		
Arguments		
const char *	File	Chemin du fichier à ouvrir.
const char *	Flag	«r» ouvre le fichier en lecture. «r+» ouvre en lecture/écriture. «w» ouvre en écriture. Tronque le fichier à une taille de zéro ou crée le fichier. «w+» ouvre en lecture/écriture. Tronque le fichier à une taille de zéro ou crée le fichier. «a» ouvre en écriture. Le fichier est crée s'il n'existe pas. «a+»ouvre en lecture/écriture. Le fichier est crée s'il n'existe pas.
Retour		
int	ID du fichier ouvert. Retourne -1 si le fichier ne peut pas être ouvert.	

IFSRead		
Lit des données depuis le fichier FileID dans le buffer Buffer.		
Arguments		
int	FileID	ID du fichier.
long long	Offset	Offset indiquant la position à partir de laquelle lire dans le fichier.
char *	Buffer	Buffer utilisé pour stocker les données lues.
int	Size	Nombre d'octets à lire.
Retour		
int	Nombre d'octets lus. Retourne -1 en cas d'erreur.	

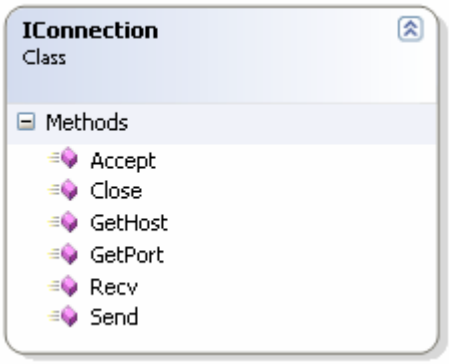
IFSWrite		
Ecrit des données à la fin du fichier FileID donné en argument.		
Arguments		
int	FileID	ID du fichier.
const char *	Buffer	Buffer contenant les données à ajouter au fichier.
int	Size	Nombre d'octet à écrire.
Retour		
int	Nombre d'octets écrits dans le fichier. Retourne -1 en cas d'erreur.	

IFSClose		
Indique au système de fichier que le module n'a plus besoin du fichier identifié par FileID.		
Arguments		
int	FileID	ID du fichier

IFSGetSize		
Donne la taille du FileID passé en argument		
Arguments		
int	FileID	ID du fichier
Retour		
long long	Taille du fichier	

IFSRemove		
Supprime le fichier File passé en argument.		
Arguments		
const char *	File	Chemin du fichier à supprimer
Retour		
bool	true si la suppression a réussie.	

4.2. Interface IConnection



Interface chargée d'implémenter les fonctions de communication. Le module n'a jamais accès à la socket serveur mais uniquement aux sockets clients.

Accept		
Permet de créer un pointeur sur une structure de donnée spécifique à la connexion rattachée à la socket. Cette fonction est appelée juste après le accept du serveur avec pour argument la socket client. Cela permet notamment de gérer les échanges de configurations initiales avec le client et de garder un pointeur sur ces données (Ex : SSL_Accept).		
Argument		
SOCKET	Sock	File descriptor
Retour		
void *	Retourne un pointeur (NULL si aucune donnée supplémentaire nécessaire)	

Close		
Permet de détruire les données crées lors de l'appel a la fonction Accept.		
Arguments		
SOCKET	Sock	File descriptor
void *	Data	Données supplémentaires

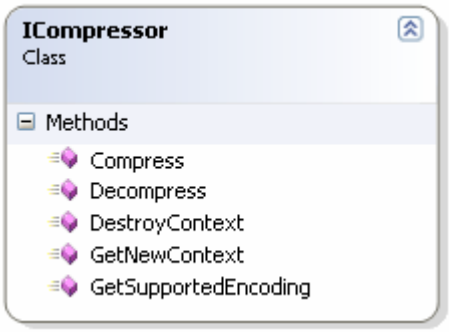
GetPort	
Renvoie le port à ouvrir en écoute par le serveur.	
Retour	
short	Numéro du port.

GetHost	
Renvoie un nom de machine ou un sous réseau sur lequel écouter. Pour écouter toutes les connexions entrantes, le module renverra « 0.0.0.0 ».	
Retour	
const char *	Host à écouter.

Recv		
Lit des données sur la socket. En cas de problème durant la lecture, la fonction renvoie -1.		
Arguments		
SOCKET	Sock	File descriptor
void *	Data	Données supplémentaires
char *	Buffer	Buffer dans lequel les données seront lues
int	Buflen	Taille du buffer
Retour		
int	Nombre d'octets lus	

Send		
Envoie des données sur la socket.		
Arguments		
SOCKET	Sock	File descriptor
void *	Data	Données supplémentaires
const char *	Buffer	Buffer de données à envoyer
int	Buflen	Taille du buffer
Retour		
bool	True si l'envoi a réussi	

4.3. Interface ICompressor



Interface chargée d'implémenter les fonctions de compression. Les modules implémentant cette interface pourront être appelés à deux moments précis :

- après le Recv pour décompresser les données en entrée.
- avant le Send pour compresser les données en sortie.

Le serveur appelle le module gérant le format demandé s'il existe. ICompressor indique les formats supportés via la méthode GetSupportedEncoding.

La taille du contenu pouvant différer entre l'entrée et la sortie du ICompressor, le serveur ne peut pas connaitre la taille que feront les données à envoyer. Le serveur doit donc spécifier dans le header soit un « Connection:close » soit un « Transfer-Encoding:chunked » en fonction de la version HTTP de la connexion.

GetNewContext		
Crée un nouveau contexte de compression.		
Arguments		
const char *	Encoding	Nom de la méthode de compression à utiliser
Retour		
void *	Pointeur sur le nouveau contexte	

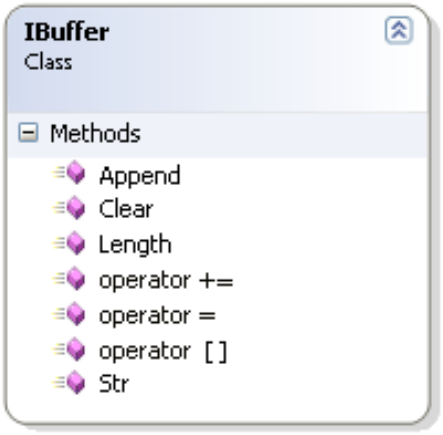
DestroyContext		
Détruit le contexte de compression.		
Arguments		
void *	Encoding	Pointeur sur le contexte à détruire

Compress		
Compresse les données du buffer en entrée et le ressort dans le buffer en sortie		
Arguments		
void *	Context	Pointeur sur le contexte de compression.
IBuffer&	In	IBuffer contenant les données à compresser
IBuffer&	Out	IBuffer contenant les données compressées
Retour		
bool	Renvoi true si la compression a réussie.	

Decompress		
Décompresse les données du buffer en entrée et les ressort dans le buffer en sortie		
Arguments		
void *	Context	Pointeur sur le contexte de décompression.
IBuffer&	In	IBuffer contenant les données à décompresser
IBuffer&	Out	IBuffer contant les données décompressées
Retour		
bool	Renvoi true si la décompression a réussie.	

GetSupportedEncoding	
Renvoie les méthodes de compression et de décompression supportées par le module	
Retour	
const char **	Tableau contenant la liste des méthodes de compression/décompression supportées

4.3.1 Interface IBuffer



Interface utilisée pour la gestion des buffers. Ces buffers sont fournis par le serveur et le module doit y lire ou écrire afin de générer sa réponse. Cette interface n'est utilisée que dans les modules « filtres » ICompressor et IStreamModifier.

Length	
Renvoi la taille du buffer.	
Retour	
int	Taille du buffer.

Str	
Renvoi un buffer contenant les données du IBuffer.	
Retour	
const char *	Contenu du buffer.

Clear
Vide le contenu du buffer.

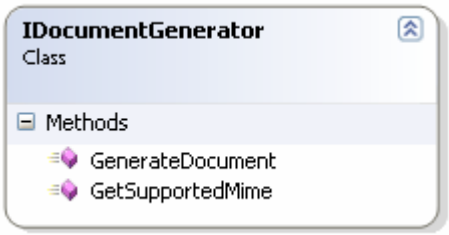
Append		
Ajoute des données à la fin du buffer.		
Arguments		
const char *	Buffer	Données
int	Size	Taille des données

operator=		
Remplace le contenu du IBuffer par un autre.		
Arguments		
IBuffer&	Right	Buffer contenant les données à copier.

operator+=		
Concatène un IBuffer dans un autre.		
Arguments		
IBuffer&	Buffer	Buffer contenant les données à concaténer.

operator[]	
Retourne un octet précis du buffer.	
Arguments	
unsigned int	Index	Index de l'octet voulu.
Retour	
char&	Référence sur l'octet pointé par l'index.

4.4. Interface IDocumentGenerator



Génération et renvoi de la réponse au client. Ce module utilise les interfaces IInput et IOuput implémentées par le serveur, et a en paramètre le chemin local du fichier demandé dans la requêtes.

Les modules implémentant cette interface génèrent du contenu Web.

Quelques exemples d'implementaton de ce module :

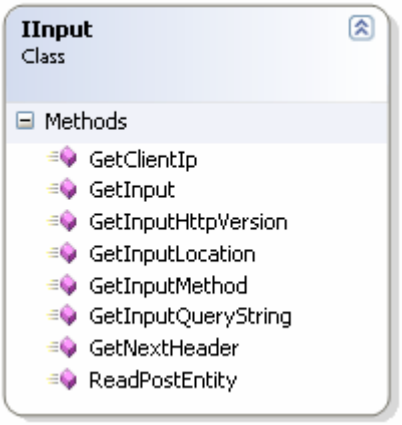
- *.php -> Exécution de scripts php.
- CGI
- Directory Browsing
- Renvoi de fichier.

Grâce à l'interface IInput le module accède à la requête client et avec l'interface IOutput il renvoie la réponse au client vers la socket.
Le serveur choisit le module à appeler en fonction des types Mime qu'il gère. Le module les indique via la fonction GetSupportedMime.

GenerateDocument		
Génère le document demandé par le client.		
Arguments		
IInput &	In	Requête du client
const char *	File	Chemin local du fichier
IOutput &	Out	Réponse à envoyer au client

GetSupportedMime	
Retourne les différents types mimes supportés.	
Retour	
const char **	Tableau contenant les types mimes.

4.4.1 Interface IInput



Cette interface permet d'accéder aux headers envoyés par le client, au chemin relatif de la page demandée ainsi qu'au corps de la requête. Les informations fournies par cette interface ne sont pas modifiables.

GetClientIp	
Retourne l'adresse IP du client.	
Retour	
int	Adresse IP

GetInput		
Renvoie la variable du header correspondant à la clé key.		
Argument		
const char *	Key	Clé
Retour		
const char *	Variable correspondant à la clé key. Retourne NULL si la clé n'existe pas.	

GetInputHttpVersion	
Retourne la version du protocole utilisé.	
Retour	
const char *	Version du protocole

GetInputLocation	
Retourne le chemin relatif du fichier ou dossier demandé.	
Retour	
const char *	Chemin relatif

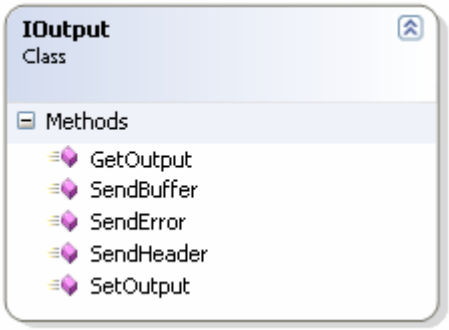
GetInputMethod	
Renvoie la méthode utilisée pour la requête (GET, POST, …)	
Retour	
const char *	Méthode

GetInputQueryString	
Retourne l'argument de la requête ou NULL si inexistant.	
Retour	
const char *	Argument de la requête

GetNextHeader		
Permet de parcourir le contenu du header. Chaque appel renvoi les pointeurs sur la paire clé/valeur suivante et renvoie true tant qu'il reste des éléments à lire.		
Arguments		
char **	Key	Nouvelle clé
char **	Value	Nouvelle valeur
Retour		
bool	Faux si le header a été entièrement parcourut.	

ReadPostEntity		
Lit le corps de la requête s'il existe (POST, PUT). A appeler en boucle tant que le nombre d'octets lus est supérieur à 0 pour récupérer l'intégralité de l'entité.		
Arguments		
char *	Rbuf	Buffer à remplir
int	Size	Taille du buffer
Retour		
int	Nombre d'octets lus.	

4.4.2 Interface IOutput



Elle permet de définir les headers et les données à envoyer au client.

GetOutput		
Renvoie un pointeur sur la valeur du header correspondant à la clé key.		
Argument		
const char *	Key	Clé
Retour		
const char *	Valeur correspondant à la clé key. Retourne NULL si la clé n'existe pas dans les headers.	

SendBuffer		
Envoie un buffer. Ce buffer passe à travers tous les « StreamModifier » puis est envoyé sur la socket client.		
Arguments		
const char *	Buffer	Buffer contenant les données à envoyer
int	Size	Taille du buffer
Retour		
bool	Retourne vrai si l'envoi à réussi, faux s'il échoue. En cas d'échec, la socket est fermée automatiquement par le serveur.	

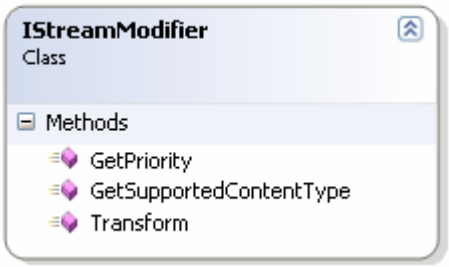
SendError		
Envoie une erreur au client. Le document generator doit quitter après avoir appelé cette fonction. Il ne doit plus envoyer de données.		
Argument		
int	Error	Numéro de l'erreur à envoyer
Retour		
bool	Retourne vrai si l'envoi à réussi	

SendHeader	
Envoie les headers. L'envoi ne peut-être fait qu'une seule fois par document généré.	
Retour	
bool	Retourne vrai si l'envoi à réussi, faux s'il a échoué. En cas d'échec, la socket est fermée automatiquement par le serveur.

SetOutput		
Définit la valeur d'une clé dans le header. Les chaines contenant la clé et la valeur sont recopiées.		
Arguments		
const char *	Key	Clé
const char *	Value	Valeur

SetStatusCode		
Définit le code de statut renvoyé au client.		
Arguments		
int	Code	Code de statut

4.5. Interface IStreamModifier



Cette interface permet de modifier les données à envoyer sans modifier le type du fichier. Cette interface n'a pas accès au header de la requête, le but de ces modules est uniquement la modification de contenu. Ces modules peuvent être chainés : utilisés les uns a la suite des autres.

La taille du contenu pouvant différer entre l'entrée et la sortie des Stream Modifier, le serveur ne peut pas connaitre la taille que feront les données à envoyer. Le serveur doit donc spécifier dans le header soit un « Connection:close » soit un « Transfer-Encoding:chunked » en fonction de la version HTTP de la connexion.

GetPriority	
Retourne la priorité du stream modifier. Les stream modifier seront utilisés dans l'ordre croissant des priorités.	
Retour	
int	Priorité

GetSupportedContentType	
Renvoie les types MIMES supportés.	
Retour	
const char **	Tableau contenant les différents types MIMES supportés par le modificateur.

Transform		
Modifie un buffer de données à envoyer.		
Arguments		
IBuffer&	In	IBuffer contenant les données à traiter.
IBuffer&	Out	IBuffer contenant les données traitées.
IBuffer&	Remain	IBuffer contenant les données non traitées.
bool	Flush	Force le renvoi du buffer.

4.5.1 Interface IBuffer

Se référer à la section 4.3.1 traitant de cette interface.

5 Annexes

5.1. Interfaces

5.1.1 IModule

```
class IModule
 {
 public:
  virtual bool          ReadConfig(const char *File) = 0;
  virtual const char    *GetModuleName() = 0;
  virtual const char    *GetModuleVersion() = 0;
  virtual void          OnLoad(IFS *Ifs) = 0;
 };
```

5.1.2 IConnection

```
class IConnection
 {
 public:
  virtual short         GetPort() = 0;
  virtual const char    *GetHost() = 0;
  virtual void          *Accept(SOCKET Socket) = 0;
  virtual int           Recv(SOCKET Socket, void *Data, char *Buffer,
                             int Buflen) = 0;
  virtual int           Send(SOCKET Socket, void *Data,
                             const char *Buffer, int Buflen) = 0;
  virtual void          Close(SOCKET Socket, void *Data) = 0;
 };
```

5.1.3 ICompressor

```
class ICompressor
 {
 public:
  virtual const char    *GetNewContext(const char *Encoding) = 0;
  virtual void          *DestroyContext(void *Encoding) = 0;
  virtual bool          Compress(void *context, IBuffer &In,
                                 IBuffer &Out) = 0;
  virtual bool          Decompress(void *context, IBuffer &In,
                                 IBuffer &Out) = 0;
  virtual const char    **GetSupportedEncoding() = 0;
 };
```

5.1.4 IDocumentGenerator

```
class IDocumentGenerator
 {
 public:
  virtual void          GenerateDocument(IInput &InHeaders,
                                         const char *File,
                                         IOutput &Out) = 0;
  virtual const char    **GetSupportedMime() = 0;
 };
```

5.1.4.1. IInput

```
class IInput
 {
 public:
  virtual const char    *GetInput(const char *Key) = 0;
  virtual const char    *GetInputMethod() = 0;
  virtual const char    *GetInputLocation() = 0;
  virtual const char    *GetInputHttpVersion() = 0;
  virtual const char    *GetInputQueryString() = 0;
  virtual int           ReadPostEntity(char *Rbuf, int Size) = 0;
  virtual bool          GetNextHeader(char **Key, char **Value) = 0;
 };
```

5.1.4.2. IOutput

```
class IOutput
 {
 public:
  virtual void          SetOutput(const char *Key,
                                  const char *Value) = 0;
  virtual const char    *GetOutput(const char *Key) = 0;
  virtual int           SendHeader() = 0;
  virtual int           SendError(int Error) = 0;
  virtual int           SendBuffer(const char *Buffer, int Size) = 0;
  virtual void          SetStatusCode(int Code) = 0;
 };
```

5.1.5 IStreamModifier

```
class IStreamModifier
 {
 public:
  virtual int           GetPriority() = 0;
  virtual const char    **GetSupportedContentType() = 0;
  virtual void          Transform(IBuffer &In, IBuffer &Out,
                              IBuffer &Remain, bool Flush = false) = 0;
 };
```

5.1.6 IFS

```
class IFS
 {
 public:
  virtual int           IFSOpen(const char *File,
                              const char Flag) = 0;
  virtual int           IFSRead(int FileID, long long Offset,
                              char *Buffer, int Size) = 0;
  virtual int           IFSWrite(int FileID, const char *Buffer,
                              int Size) = 0;
  virtual void          IFSClose(int FileID) = 0;
  virtual long long     IFSGetSize(int FileID) = 0;
  virtual bool          IFSRemove(const char *File) = 0;
 };
```

5.1.7 IBuffer

```
class IBuffer
 {
 public:
  virtual int           Length() = 0;
  virtual const char    *Str() = 0;
  virtual void          Clear() = 0;
  virtual void          Append(const char *Buffer, int Size) = 0;
  virtual IBuffer&      operator=(IBuffer &Right) = 0;
  virtual IBuffer&      operator+=(IBuffer &Right) = 0;
  virtual char&         operator[](unsigned int Index) = 0;
 };
```

5.2. Exemples de modules

5.2.1 IModule

```
// Le extern "C" indique au compilateur de ne pas modifier le nom de
notre fonction GetNewInstance
// Le __declspec(dllexport) est utilise par Visual Studio afin que la
fonction GetNewInstance soit accessible dans la dll.
//    Cette option est obligatoire pour Visual Studio. Elle ne doit pas
etre presente pour les autres compilateurs.

#ifdef WIN32
# define DLLEXPORT      __declspec(dllexport)
#else
# define DLLEXPORT
#endif

extern "C" DLLEXPORT IModule  *GetNewInstance();


class MyModule : public IModule
{
      // Implementation de l'interface IModule
      // Toutes les fonctions virtuelles pures d'IModule doivent etre
implementees
      public:
            bool        ReadConfig(char *file);
            string      GetModuleName();
            string      GetModuleVersion();
            void        OnLoad(IFS *Ifs);

      // Notre classe
      public:
            MyModule();
            ~MyModule();

      private:
            IFS         *_ifs;
}

// Cette fonction est obligatoire pour avoir un module valide
// Retour: une instance de notre module
extern "C" DLLEXPORT IModule  *GetNewInstance()
{
      return (new MyModule());
}
```

```
// ---------------------------------
// Implementation d'IModule
// ---------------------------------

// A chaque appel, cette fonction lis le fichier de configuration
associe au module
//    Apres le chargement du module (GetNewInstance), un appel a
ReadConfig est effectue
// Retour: false s'il y a eu un probleme durant la configuration du
module
//          true si la configuration s'est effectuee correctement
bool  MyModule::ReadConfig(char *file)
{
      // On peut ici lire le fichier file fournit en parametre et
recuperer la configuration du module
      // Le format de ce fichier est definit par le createur du module

      // Notre module n'a pas besoin d'informations supplementaires
pour se configurer, la configuration s'est donc bien passee
      return (true);
}

// Retour: le nom du module
string      MyModule::GetModuleName()
{
      return ("MyModule");
}

// Retour: la version du module
string      MyModule::GetModuleVersion()
{
      return ("0.42b");
}

void  MyModule::OnLoad(IFS *Ifs)
{
      this->ifs = Ifs;
}

// ------------------------------------
// Methodes propres a la classe
// ------------------------------------

// Constructeur de la classe
// On peut ici creer tous ce dont le module a besoin et qui n'a pas
besoin de configuration
MyModule::MyModule()
{
}

// Destructeur de la classe
// On doit ici liberer toutes les ressources utilisees par le module
MyModule::~MyModule()
{
}
```

5.2.2 IConnection

```
// L'implementation de l'interface IConnection permet au module de
fournir le support d'un protocole utilisant TCP dans lequel sera
encapsule le protocole HTTP
class MyConnection : public IModule, public IConnection
{
      // Implementation d'IModule
      public:
            bool        ReadConfig(char *file);
            string      GetModuleName();
            string      GetModuleVersion();
            void        OnLoad(IFS *Ifs);

      // Implementation d'IConnection
      public:
            short       GetPort();
            char        *GetHost();
            void        *Accept(SOCKET Socket);
            int         Recv(SOCKET Socket, void *Data, char *Buffer,
                              int Buflen);
            int         Send(SOCKET Socket, void *Data,
                              const char *Buffer, int Buflen);
            void        Close(SOCKET Socket, void *Data);

      // Notre classe
      public:
            MyConnection();
            ~MyConnection();

      private:
            IFS         *_ifs;
            short       _port;
            char        *_host;
}

// ---------------------------------
// Implementation d'IModule
// ---------------------------------

bool  MyConnection::ReadConfig(char *file)
{
      this->_port = 8080;
      this->_host = "0.0.0.0";
      return (true);
}

string      MyConnection::GetModuleName()
{
      return ("MyConnection");
}

string      MyConnection::GetModuleVersion()
{
      return ("0.0");
```

```
}

void  MyConnection::OnLoad(IFS *Ifs)
{
      this->_ifs = Ifs;
}

// ----------------------------------------
// Implementation d'IConnection
// ----------------------------------------

// Retour: le port a utiliser pour la connexion
short MyConnection::GetPort()
{
      return (this->_port);
}

// Retour: l'host pour lequel la connexion doit ecouter
char  *MyConnection::GetHost()
{
      return (this->_host);
}

// Permet de realiser, apres l'accept du serveur, un second accept
grace auquel on pourra initialiser le protocole de transport
// Retour: un pointeur sur des informations supplementaires pour la
connexion
void  *MyConnection::Accept(SOCKET Socket)
{
      // Nous n'avons pas besoin d'effectuer le deuxieme accept, ni de
donnees supplementaires, on renvoit un pointeur null
      return (NULL);
}

// Recoit de donnees du client. L'argument Data fournit les donnees
supplementaires venant du Accept
// Retour: le nombre d'octets lus
int         MyConnection::Recv(SOCKET Socket, void *Data, char *Buffer,
int Buflen)
{
      return (recv(Socket, Buffer, Buflen));
}

// Envoi des donnees au client. L'argument Data fournit les donnees
supplementaires venant du Accept
// Retour: le nombre d'octets envoyes
int         MyConnection::Send(SOCKET Socket, void *Data, char *Buffer,
int Buflen)
{
      return (send(Socket, Buffer, Buflen));
}

// Appele a la fermeture de la connexion. Permet de detruire les
donnees supplementaires venant du Accept
void  MyConnection::Close(SOCKET Socket, void *Data)
{
}
```

```
// ------------------------------------
// Methodes propres a la classe
// ------------------------------------

MyConnection::MyConnection()
{
}

MyConnection::~MyConnection()
{
}
```

5.2.3 ICompressor

```
// L'implementation de l'interface ICompressor permet au module de
decompresser les donnees envoyees par le client et de compresser les
donnees a lui envoyer
class MyCompressor : public IModule, public ICompressor
{
      // Implementation d'IModule
      public:
            bool        ReadConfig(char *file);
            string      GetModuleName();
            string      GetModuleVersion();
            void        OnLoad(IFS *Ifs);

      // Implementation d'ICompressor
      public:
            bool        Compress(const char *Encoding, IBuffer &In,
                              IBuffer &Out);
            bool        Decompress(const char *Encoding, IBuffer &In,
                              IBuffer &Out);
            const char  **GetSupportedEncoding();

      // Notre classe
      public:
            MyCompressor();

      private:
            IFS         *_ifs;
}

// ---------------------------------
// Implementation d'IModule
// ---------------------------------

bool  MyCompressor::ReadConfig(char *file)
{
      return (true);
}

string      MyCompressor::GetModuleName()
{
      return ("MyCompressor");
}

string      MyCompressor::GetModuleVersion()
{
      return ("0.1");
}

void  MyCompressor::OnLoad(IFS *Ifs)
{
      this->_ifs = Ifs;
}
```

```
// ---------------------------------------
// Implementation d'ICompressor
// ---------------------------------------

// Compresse le buffer donne en entree et le renvoit grace a la
variable OutBuffer
// L'algorithme de compression a utiliser est indique par la variable
Encoding
// Retour: succes ou echec de la compression
bool        MyCompressor::Compress(const char *Encoding, IBuffer &In,
IBuffer &Out)
{
      Out = In;
      return (true);
}

// Decompresse le buffer donne en entree et le renvoit grace a la
variable OutBuffer
// L'algorithme de decompression a utiliser est indique par la variable
Encoding
// Retour: succes ou echec de la decompression
bool        MyCompressor::Decompress(const char *Encoding, IBuffer &In,
IBuffer &Out)
{
      Out = In;
      return (true);
}

// Specifie quels type d'encodage est fournit par le module. Un module
peut supporter plusieurs encodages differents.
// Retour: un tableau contenant le nom de chacun des algorithmes
fournis par le module.
const char  **MyCompressor::GetSupportedEncoding()
{
      char  *encoding[2] = {"Identity", NULL};

      return (encoding);
}

// -------------------------------------
// Methodes propres a la classe
// -------------------------------------

MyCompressor::MyCompressor()
{
}
```

5.2.4 IDocumentGenerator

```
// L'implementation de l'interface IDocumentGenerator permet de generer
une page demandee par le client
class MyGenerator : public IModule, public IDocumentGenerator
{
      // Implementation d'IModule
      public:
            bool        ReadConfig(char *file);
            string      GetModuleName();
            string      GetModuleVersion();
            void        OnLoad(IFS *Ifs);

      // Implementation d'IDocumentGenerator
      public:
            void        GenerateDocument(IInput &InHeaders,
                              const char *File, IOutput &Out);
            char        **GetSupportedMime();

      // MyGenerator
      public:
            MyGenerator();

      private:
            IFS         *_ifs;
}

// ---------------------------------
// Implementation d'IModule
// ---------------------------------

bool  MyGenerator::ReadConfig(char *File)
{
      return (true);
}

string      MyGenerator::GetModuleName()
{
      return ("MyGenerator");
}

string      MyGenerator::GetModuleVersion()
{
      return ("0.2");
}

void  MyGenerator::OnLoad(IFS *Ifs)
{
      this->_ifs = Ifs;
}
```

```
// ------------------------------------------------
// Implementation d'IDocumentGenerator
// ------------------------------------------------

// Genere le document file demande par le client.
// Les informations relatives a la requete emise par le client se
trouvent dans In
// Les informations relatives a la reponse se trouvent dans Out
void  MyGenerator::GenerateDocument(IInput &In, const char *File,
                                    IOutput &out)
{
      string            output;
      ostringstream     oss;

      output = "<html><body>Test<br />";
      output += In.GetInputMethod();
      output += "</body></html>";
      oss << output.length();
      Out.SetOutput("Content-Length", (char *)oss.str().c_str());
      if (Out.SendHeader() <= 0)
            return;
      Out.SendBuffer((char *)output.c_str(), output.length());
}

// Retour: tableau contenant les types mimes supportes par ce
generateur de document
char  **MyGenerator::GetSupportedMime()
{
      char  *mime[2] = {"application/test", NULL};

      return (mime);
}

// -----------------
// MyGenerator
// -----------------

MyGenerator::MyGenerator()
{
}
```

5.2.5 IStreamModifier

```
// L'implementation de la classe IStreamModifier permet de modifier les
donnees generees juste avant de les envoyer au client
// Cette classe ne peut pas modifier le header de la reponse, le type
du document reste le meme.
class MyStreamModifier : public IModule, public IStreamModifier
{
      // Implementation d'IModule
      public:
            bool        ReadConfig(char *File);
            string      GetModuleName();
            string      GetModuleVersion();
            void        OnLoad(IFS *Ifs);

      // Implementation d'IStreamModifier
      public:
            void        Transform(IBuffer &In, IBuffer &Out,
                                  IBuffer &Remain, bool Flush);
            char        **GetSupportedContentType();

      // MyStreamModifier
      public:
            MyStreamModifier();

      private:
            IFS         *_ifs;
}

// ---------------------------------
// Implementation d'IModule
// ---------------------------------

bool  MyStreamModifier::ReadConfig(char *File)
{
      return (true);
}

string      MyStreamModifier::GetModuleName()
{
      return ("MyStreamModifier");
}

string      MyStreamModifier::GetModuleVersion()
{
      return ("0.0.1");
}

void  MyStreamModifier::OnLoad(IFS *Ifs)
{
      this->_ifs = Ifs;
}
```

```
// ---------------------------------------------
// Implementation d'IStreamModifier
// ---------------------------------------------

// Recoit un buffer In, le traite et renvoit les donnees traitees dans
le buffer Out.
// Toutes les donnees ne pouvant etre traitees sont stockees dans le
buffer Remain.
//          Ce buffer est gere par le serveur.
// Si le flag Flush est a true, le module est force de retourner son
buffer Remain
//          (tout en faisant le traitement, si possible, sur le buffer
Remain et le buffer In)
void  MyStreamModifier::Transform(IBuffer &In, IBuffer &Out, IBuffer
&Remain, bool Flush)
{
      Out = In;
      for (int i = 0; i < Out.Length(); i++)
            if (Out[i] == 'z')
                  Out[i] = 'a';
}

// Retour: tableau contenant les ContentTypes supportes par le module
char  **MyStreamModifier::GetSupportedContentType()
{
      char  *content[2] = {"text/test", NULL};

      return (content);
}
```

5.2.6 CBuffer

Exemple d'implémentation de l'interface IBuffer dans le serveur.

```
#include <string>

using namespace std;

class CBuffer : public IBuffer
{
private:
      string      _str;
public:
      size_t      Length();
      const char  *Str();
      void        Clear();
      void        Append(const char *Buffer, int Size);
      IBuffer&    operator=(IBuffer &Right);
      IBuffer&    operator+=(IBuffer &Right);
      char&       operator[](unsigned int Index);
};

int   CBuffer::Length()
{
      return ((int)this->_str.length());
}

const char  *CBuffer::Str()
{
      return (this->_str.c_str());
}

void  CBuffer::Clear()
{
      this->_str.clear();
}

void  CBuffer::Append(const char *Buffer, int Size)
{
      this->_str.append(string(Buffer, Size));
}

IBuffer&    CBuffer::operator =(IBuffer &Right)
{
      this->_str = string(Right.Str(), Right.Length());
      return (*this);
}

IBuffer&    CBuffer::operator +=(IBuffer &Right)
{
      this->_str.append(string(Right.Str(), Right.Length()));
      return (*this);
}
```

```
char& CBuffer::operator [](unsigned int Index)
{
	return (this->_str[Index]);
}
```

5.3. Schéma détaillé du comportement serveur