

DOSSIER DE DEVELOPPEMENT DU PROJET lightREST3

15/01/2026

Partie 1

Projet

Projet

Code

Initialisation de lightREST3

```
<COMPILE SI Configuration>"LightREST Linux">
  CONSTANCE
    DLL64_NAME    = "goREST64.dll"
    DLL32_NAME    = "goREST32.dll"

    //DLL path on my development machine
    DEV64_DLL     = "C:\projets\goREST\goREST64.dll"
    DEV_DLL32     = "C:\projets\goREST\goREST32.dll"
  FIN
<SINON>
  CONSTANCE
    DLL64_NAME    = "goREST.so"
    DEV_DLL       = ""
  FIN
<FIN>
```

Projet

Statistiques sur le code

	Lignes ¹	% comm. ¹	Lig./trait. ¹
FEN	548	66	49
FEN_DLL	32	41	4
lightREST3	15	7	15
lrDLL	481	14	22
lrDLLLinux	11	36	2
lrDLLWindows32	11	36	2
lrDLLWindows64	11	36	2
lrRequest	217	53	21
lrResponse	346	56	34
lrRoute	115	1	23
lrServer	821	7	13
lrSession	488	13	14
	2879	31	16

¹ Lignes : Nombre total de lignes de code.

% comm. : Pourcentage de commentaires dans le code.

Lig./trait. : Nombre de lignes de code par traitement.

Partie 2

Classe

IrResponse

Code

Déclaration de IrResponse

```

IrResponse est Classe
  Body      est Buffer
  Status     est entier
  ContentType est chaîne
  Headers    est tableau associatif (ccSansCasse+AvecDoublon) de chaîne
  IsBinary  est booléen
FIN

// Déclaration de constantes spécifique
// Specific constant declaration
<BLOC const>
  CONSTANTE
    //Content Types
    ContentJSON      = "application/json"
    ContentBIN       = "application/octet-stream"
    ContentBMP       = "image/bmp"
    ContentCSV       = "text/csv"
    ContentGIF       = "image/gif"
    ContentHTML      = "text/html"
    ContentJPG       = "image/jpeg"
    ContentPNG       = "image/png"
    ContentWEBP      = "image/webp"
    ContentSVG       = "image/svg+xml"
    ContentPDF       = "application/pdf"
    ContentTIFF      = "image/tiff"
    ContentTXT       = "text/plain"
    ContentXML       = "application/xml"
    ContentZIP       = "application/zip"

    //HTTP STATUS
    StatusContinue   = 100 // RFC 7231, 6.2.1
    StatusSwitchingProtocols = 101 // RFC 7231, 6.2.2
    StatusProcessing  = 102 // RFC 2518, 10.1
    StatusEarlyHints  = 103 // RFC 8297

    StatusOK         = 200 // RFC 7231, 6.3.1
    StatusCreated     = 201 // RFC 7231, 6.3.2
    StatusAccepted    = 202 // RFC 7231, 6.3.3
    StatusNonAuthoritativeInfo = 203 // RFC 7231, 6.3.4
    StatusNoContent   = 204 // RFC 7231, 6.3.5
    StatusResetContent = 205 // RFC 7231, 6.3.6
    StatusPartialContent = 206 // RFC 7233, 4.1
    StatusMultiStatus = 207 // RFC 4918, 11.1
    StatusAlreadyReported = 208 // RFC 5842, 7.1
    StatusContentDifferent = 210 // RFC 3229, 10.4.1
    StatusIMUsed       = 226 // RFC 3229, 10.4.1

    StatusMultipleChoices = 300 // RFC 7231, 6.4.1
    StatusMovedPermanently = 301 // RFC 7231, 6.4.2
    StatusFound           = 302 // RFC 7231, 6.4.3
    StatusSeeOther        = 303 // RFC 7231, 6.4.4
    StatusNotModified     = 304 // RFC 7232, 4.1
    StatusUseProxy        = 305 // RFC 7231, 6.4.5
    StatusTemporaryRedirect = 307 // RFC 7231, 6.4.7

```

```

StatusPermanentRedirect          = 308 // RFC 7538, 3
StatusTooManyRedirects           = 310 // RFC 7538, 3

StatusBadRequest                 = 400 // RFC 7231, 6.5.1
StatusUnauthorized               = 401 // RFC 7235, 3.1
StatusPaymentRequired           = 402 // RFC 7231, 6.5.2
StatusForbidden                 = 403 // RFC 7231, 6.5.3
StatusNotFound                  = 404 // RFC 7231, 6.5.4
StatusMethodNotAllowed          = 405 // RFC 7231, 6.5.5
StatusNotAcceptable             = 406 // RFC 7231, 6.5.6
StatusProxyAuthRequired         = 407 // RFC 7235, 3.2
StatusRequestTimeout            = 408 // RFC 7231, 6.5.7
StatusConflict                  = 409 // RFC 7231, 6.5.8
StatusGone                      = 410 // RFC 7231, 6.5.9
StatusLengthRequired            = 411 // RFC 7231, 6.5.10
StatusPreconditionFailed        = 412 // RFC 7232, 4.2
StatusRequestEntityTooLarge     = 413 // RFC 7231, 6.5.11
StatusRequestURITooLong        = 414 // RFC 7231, 6.5.12
StatusUnsupportedMediaType      = 415 // RFC 7231, 6.5.13
StatusRequestedRangeNotSatisfiable = 416 // RFC 7233, 4.4
StatusExpectationFailed        = 417 // RFC 7231, 6.5.14
StatusTeapot                   = 418 // RFC 7168, 2.3.3
StatusPageExpired               = 419 // RFC 7168, 2.3.3
StatusBadMapping                = 421 // RFC 4918, 11.2
StatusUnprocessableEntity       = 422 // RFC 4918, 11.2
StatusLocked                    = 423 // RFC 4918, 11.3
StatusMethodFailure             = 424 // RFC 4918, 11.4
StatusTooEarly                  = 425 // RFC 8470, 5.2.
StatusUpdateRequired            = 426 // RFC 7231, 6.5.15
StatusInvalidDigitalSignature   = 427 // RFC 7231, 6.5.15
StatusPreconditionRequired      = 428 // RFC 6585, 3
StatusTooManyRequests           = 429 // RFC 6585, 4
StatusRequestHeaderFieldsTooLarge = 431 // RFC 6585, 5
StatusRetryWith                 = 449 // RFC 7725, 3
StatusBlockedByWParentalControls = 450 // RFC 7725, 3
StatusUnavailableForLegalReasons = 451 // RFC 7725, 3
StatusUnrecoverableError       = 456 // RFC 7725, 3

StatusInternalServerError       = 500 // RFC 7231, 6.6.1
StatusNotImplemented           = 501 // RFC 7231, 6.6.2
StatusBadGateway               = 502 // RFC 7231, 6.6.3
StatusServiceUnavailable       = 503 // RFC 7231, 6.6.4
StatusGatewayTimeout           = 504 // RFC 7231, 6.6.5
StatusHTTPVersionNotSupported  = 505 // RFC 7231, 6.6.6
StatusVariantAlsoNegotiates    = 506 // RFC 2295, 8.1
StatusInsufficientStorage      = 507 // RFC 4918, 11.5
StatusLoopDetected             = 508 // RFC 5842, 7.2
StatusBandwidthLimitExceeded   = 509
StatusNotExtended              = 510 // RFC 2774, 7
StatusNetworkAuthenticationRequired = 511 // RFC 6585, 6

```

FIN

<FIN>

Méthode __IsItemMemo

```

// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//[ <Résultat> = ] __IsItemMemo (<pType> est entier)
//
// Paramètres :
// pType (entier) : <indiquez ici le rôle de pType>
// Valeur de retour :
// booléen : <indiquez ici les valeurs possibles ainsi que leur interprétation>

```

```
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCÉDURE GLOBAL PRIVÉE __IsItemMemo ( LOCAL pType est entier ) : booléen

RENVOYER pType _DANS_ (hRubMémoBinaire) //,hRubMémoBinaire4,hRubMémoTexte,hRubMémoUnicode
```

Méthode __ListItem

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//[ <Résultat> = ] __ListItem (<pSource> est objet source de données [, <pColumns> est chaîne])
//
// Paramètres :
// pSource (objet source de données) : <indiquez ici le rôle de pSource>
// pColumns (chaîne ANSI - valeur par défaut="*") : <indiquez ici le rôle de pColumns>
// Valeur de retour :
// multi-valeur : // Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCÉDURE GLOBALE PRIVÉE __ListItem(pSource est Source de Données, LOCAL pColumns est chaîne="*") : (
tableau associatif d'entiers, booléen)

sRubs          est chaîne    = HListeRubrique(pSource, hLstDétail)
sRub           est chaîne
sNomRub        est chaîne
taRubs         est tableau associatif (ccSansCasse) d'entiers
bTypeMemoPresent est booléen

POUR TOUTE CHAÎNE sRub DE sRubs SÉPARÉE PAR CRLF
  sNomRub = ExtraitChaîne(sRub, 1)
  SI pColumns = "*" _OU_ Contient(", "+pColumns+",", ", "+sNomRub+",") ALORS
    taRubs[sNomRub] = Val(ExtraitChaîne(sRub, 3))
    SI ::__IsItemMemo(taRubs[sNomRub]) ALORS
      bTypeMemoPresent = Vrai
  FIN
FIN

RENVOYER (taRubs,bTypeMemoPresent)
```

Méthode __MemoToVariant

```
PROCÉDURE PRIVÉE GLOBALE __MemoToVariant(pTable, LOCAL pMemoRub est chaîne)

vBase64      est Variant
sInfoMemo    est chaîne
sFichier     est chaîne
iImage       est Image

SI pMemoRub <> "" ALORS
  sInfoMemo = HInfoMémo(pTable, pMemoRub)

  SI Majuscule(ExtraitChaîne(sInfoMemo, 1)) _DANS_ ("BIN","IMG") ALORS
    sFichier = ExtraitChaîne(sInfoMemo, 2)

    iImage = {"pTable."+pMemoRub, indVariable}

    vBase64.Name = fExtraitChemin(sFichier, fFichier)
    SI vBase64.Name = "" ALORS
      vBase64.Name = "image"
  FIN
```

```

    SELON fExtraitChemin(sFichier, fExtension).Minuscule()
    CAS ".jpg", ".jpeg"
        vBase64.Content = Encode(dSauveImageJPEG(iImage, enMémoire), encodeBASE64SansRC)
        vBase64.Name += ".jpg"
    CAS ".bmp"
        vBase64.Content = Encode(dSauveImageBMP(iImage, enMémoire), encodeBASE64SansRC)
        vBase64.Name += ".bmp"
    CAS ".gif"
        vBase64.Content = Encode(dSauveImageGIF(iImage, enMémoire), encodeBASE64SansRC)
        vBase64.Name += ".gif"
    AUTRES CAS
        vBase64.Content = Encode(dSauveImagePNG(iImage, enMémoire), encodeBASE64SansRC)
        vBase64.Name += ".png"
    FIN
FIN
FIN

RENNVOYER vBase64

```

Constructeur

```
PROCÉDURE Constructeur()
```

Destructeur

```
PROCÉDURE Destructeur()
```

Méthode RecordToVariant

```

// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'RecordToVariant' :
//
// Syntaxe :
// [ <Résultat> = ] RecordToVariant (<pSource> est objet source de données [, <pColumns> est chaîne])
//
// Paramètres :
// pSource (objet source de données) : <indiquez ici le rôle de pSource>
// pColumns (chaîne ANSI - valeur par défaut="") : <indiquez ici le rôle de pColumns>
// Valeur de retour :
// variant : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
// Déterminez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION GLOBALE RecordToVariant(pSource est Source de Données, LOCAL pColumns est chaîne="")

vResult          est Variant
vRecords          est Variant
vRecord           est Variant
taRubs            est tableau associatif (ccSansCasse) d'entiers
bTypeMemoPresent  est booléen
sRub              est chaîne
eType             est entier

(taRubs,bTypeMemoPresent) = ::__ListItem(pSource,pColumns)

```

```

SI bTypeMemoPresent _OU_ pColumns <> "*" ALORS
  POUR TOUT eType, sRub de taRubs
    SI __IsItemMemo(eType) ALORS
      {"vResult."+sRub, indVariable} = __MemoToVariant(pSource, sRub)
    SINON
      {"vResult."+sRub, indVariable} = {"pSource."+sRub, indVariable}
    FIN
  FIN
SINON
  //Pas besoin de filtrer les colonnes, ni d'encoder les mémos : on va au plus rapide
  vRecords = JSONVersVariant(HEnregistrementVersJSON(pSource))
  POUR TOUT vRecord DE vRecords
    vResult = vRecord
  SORTIR
FIN
FIN

RENOYER vResult

```

Méthode SetBinaryContent

```

// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'SetBinaryContent' :
//
// Syntaxe :
//SetBinaryContent (<pContent> est buffer [, <pContentType> est chaîne])
//
// Paramètres :
//  pContent (buffer) : <indiquez ici le rôle de pContent>
//  pContentType (chaîne ANSI - valeur par défaut="") : <indiquez ici le rôle de pContentType>
// Valeur de retour :
//  Aucune
//
// Notes :
// Décrivez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCÉDURE SetBinaryContent(pContent est Buffer, pContentType est chaîne="")

//:Body = encode(pContent, encodeBASE64SansRC)
:IsBinary = Vrai
:Body = pContent

SI pContentType<>" " ALORS
  :ContentType = pContentType
FIN

```

Méthode SetHeaderValue

```

// Résumé : Affecte une valeur à une entête (HEADER) de la réponse REST
// Description des paramètres d'entrée/sortie de 'SetHeaderValue' :
//
// Syntaxe :
//SetHeaderValue (<pTag> est chaîne, <pValue> est chaîne)
//
// Paramètres :
//  pTag (chaîne ANSI) : Nom de l'entête
//  pValue (chaîne ANSI) : Valeur
// Valeur de retour :
//  Aucune

```

```
//
// Notes :
// Détaillez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
// oResponse.SetHeaderValue ("TOKEN", GetUUID())
//
//
// Summary : Assigns a value to a HEADER of the REST response
// Description of input/output parameters of 'SetHeaderValue' :
//
// Syntax:
// SetHeaderValue (<pTag> is string, <pValue> is string)
//
// Parameters :
//   pTag (string) : Header
//   pValue (string) : Value
//
// Return value:
//   None
//
// Example :
// oResponse.SetHeaderValue ("TOKEN", GetUUID())
PROCÉDURE SetHeaderValue(LOCAL pTag est chaîne, LOCAL pValue est chaîne)

:Headers[pTag] = pValue
```

Méthode SourceToVariant

```
// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'SourceToVariant' :
//
// Syntaxe :
//[<Résultat> = ] SourceToVariant (<pSource> est objet source de données [, <pColumns> est chaîne])
//
// Paramètres :
//   pSource (objet source de données) : <indiquez ici le rôle de pSource>
//   pColumns (chaîne ANSI - valeur par défaut="*") : <indiquez ici le rôle de pColumns>
// Valeur de retour :
//   variant : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
// Détaillez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION GLOBALE SourceToVariant(pSource est Source de Données, LOCAL pColumns est chaîne="*")

tvData          est tableau de Variant
vResult         est Variant
vRecord         est Variant
vArrayResult    est tableau de Variant
taRubs          est tableau associatif (ccSansCasse) d'entiers
bTypeMemoPresent est booléen
sRub            est chaîne
eType           est entier

(taRubs,bTypeMemoPresent) = ::__ListItem(pSource,pColumns)

// Parcoure les données
POUR TOUT {pSource..Nom, indFichier}
```

```
SI bTypeMemoPresent _OU_ pColumns <> "*" ALORS
  VariableRAZ(vRecord)
  POUR TOUT eType, sRub de taRubs
    SI ::__IsItemMemo(eType) ALORS
      {"vRecord."+sRub, indVariable} = ::__MemoToVariant(pSource, sRub)
    SINON
      {"vRecord."+sRub, indVariable} = {"pSource."+sRub, indVariable}
    FIN
  FIN
  vArrayResult.ajoute(vRecord)
SINON
  //Pas besoin de filtrer les colonnes, ni d'encoder les mémos : on va au plus rapide
  tvData.Ajoute(::RecordToVariant(pSource))
FIN

// Formate le retour
SI vArrayResult..occurrence > 0 ALORS
  vResult = vArrayResult
SINON
  vResult = tvData
FIN

RENVOYER vResult
```

IrRequest

Code

Déclaration de IrRequest

```

IrRequest est Classe
  ID                est chaîne
  Body              est chaîne
  Method            est chaîne
  URL               est lrURL
  Proto             est chaîne
// ProtoMajor       est entier
// ProtoMinor       est entier
  Header            est tableau associatif (ccSansCasse+AvecDoublon) de chaînes
  ContentLength     est entier sur 8 octets
// TransferEncoding est tableau de chaînes
// Close            est booléen
  Host              est chaîne
// Form             est tableau associatif (ccSansCasse) de chaînes
  PostForm          est tableau associatif (ccSansCasse) de chaînes
// MultipartForm    est lrForm, serialise = faux
  Files             est tableau associatif (ccSansCasse) de lrFile
// Trailer          est tableau associatif (ccSansCasse) de chaînes
  RemoteAddr        est chaîne
  RequestURI        est chaîne
  Authentication    est lrAuthentication
// TLS:             est r.TLS
  Vars              est tableau associatif (ccSansCasse) de chaînes

  CustomData        est Variant
  ServerCustomData  est Variant
  RouteCustomData   est Variant
FIN

// Déclaration de type spécifique nécessaire dans la classe
// Specific type declaration needed in the class
<BLOC types>
  lrURL est Structure
    Scheme          est chaîne
    // Opaque        est chaîne
    User            est chaîne
    Host            est chaîne
    Path            est chaîne
    RawPath         est chaîne
    QueryValues     est tableau associatif (ccSansCasse) de chaînes
    // ForceQuery    est booléen
    // RawQuery      est chaîne
    // Fragment      est chaîne
    // RawFragment   est chaîne
  FIN

  //lrForm est structure
  // Value          est tableau associatif (ccSansCasse) de chaînes
  // File           est tableau associatif (ccSansCasse) de chaînes
  //FIN

  lrFile est Structure
    Name est chaîne
    Content est Buffer

```

```

FIN

  lrAuthentication est Structure
    User      est chaîne
    Password  est chaîne
  FIN
<FIN>

```

Méthode __IsItemMemo

```

FONCTION GLOBAL PRIVÉE __IsItemMemo( LOCAL pType est entier ) : booléen

RENVOYER pType _DANS_ (hRubMémoBinaire) //,hRubMémoBinaire4,hRubMémoTexte,hRubMémoUnicode

```

Méthode __ListItem

```

// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
// [ <Résultat> = ] __ListItem (<pSource> est objet source de données [, <pColumns> est chaîne])
//
// Paramètres :
//   pSource (objet source de données) : <indiquez ici le rôle de pSource>
//   pColumns (chaîne ANSI - valeur par défaut="*") : <indiquez ici le rôle de pColumns>
// Valeur de retour :
//   multi-valeur : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION GLOBALE PRIVÉE __ListItem(pSource est Source de Données, LOCAL pColumns est chaîne = "*") : (
tableau associatif d'entiers, booléen)

sRubs      est chaîne
sRub       est chaîne
sNomRub    est chaîne
taRubs     est tableau associatif (ccSansCasse) d'entiers
bTypeMemoPresent est booléen

sRubs = HListeRubrique(pSource, hLstDetail)

POUR TOUTE CHAÎNE sRub DE sRubs SÉPARÉE PAR CRLF
  SI sRub = "" ALORS CONTINUER

  sNomRub = ExtraireChaîne(sRub, 1)

  SI sNomRub <> "" _ET_ (pColumns = "*" _OU_ Contient(", "+pColumns+",", ", "+sNomRub+",")) ALORS
    taRubs[sNomRub] = Val(ExtraireChaîne(sRub, 3))
    SI ::__IsItemMemo(taRubs[sNomRub]) ALORS
      bTypeMemoPresent = Vrai
    FIN
  FIN
FIN

RENVOYER (taRubs, bTypeMemoPresent)

```

Constructeur

```
PROCÉDURE Constructeur()
```

Destructeur

PROCÉDURE Destructeur()

Méthode GetHeaderSubValue

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
// [ <Résultat> = ] GetHeaderSubValue (<pHeader> est chaîne, <pSubHeader> est chaîne)
//
// Paramètres :
// pHeader (chaîne ANSI) : <indiquez ici le rôle de pHeader>
// pSubHeader (chaîne ANSI) : <indiquez ici le rôle de pSubHeader>
// Valeur de retour :
// chaîne ANSI : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE GetHeaderSubValue(LOCAL pHeader est chaîne, LOCAL pSubHeader est chaîne) : chaîne
cRes est chaîne
ePos est entier

POUR TOUT value DE :Header = pHeader
  // SubHeader est égale ou commence par la valeur
  // SubHeader is equal or start with the value
  SI (pSubHeader ~= value _OU_ pSubHeader.Minuscule()+"=" [= value.Minuscule()]) ALORS
    ePos = Position(value, "=")
    SI ePos > 0 ALORS
      RENVOYER value[[ePos+1 À]]
    SINON
      RENVOYER value
  FIN
FIN
FIN

RENOYER cRes
```

Méthode GetHeaderValue

```
// Résumé : Renvoie la valeur d'un header de la requête REST
// Description des paramètres d'entrée/sortie de 'GetHeaderValue' :
//
// Syntaxe:
// GetHeaderValue(Header est chaîne)
//
// Paramètres :
// <header> Nom du header
// Valeur de retour:
// Valeur du header ou chaîne vide si inexistant
//
// Exemple :
// SesssionID = oRequest.GetHeaderValue("session")
//
//
// Summary : Returns a header value of the REST request
// Description of input/output parameters of 'GetHeaderValue' :
//
// Syntaxe :
// [[ <Résultat> = ] GetHeaderValue (<pHeader> est chaîne)
//
// Parameters :
// <header> Header name
```

```
// Valeur de retour / Return value:
// Header value or empty string if non-existent
//
// Example :
// SessionID = oRequest:GetHeaderValue("session")
FUNCTION PUBLIQUE GetHeaderValue(LOCAL pHeader est chaîne) : chaîne

//La tableau associatif est avec doublons, donc nécessité de le parcourir pour trouver la clé recherchée
//The associative table has duplicates, so it is necessary to browse it to find the searched key.

SI pHeader <> "" _ET_ :Header[pHeader]..existe ALORS
    POUR TOUT value DE :Header = pHeader
        RENVOYER (value = "" ? pHeader SINON value)
    FIN
FIN

RENVOYER ""
```

Méthode GetRouteValue

```
// Résumé : Renvoie la valeur d'une variable de la route REST
// Description des paramètres d'entrée/sortie de 'GetRouteValue' :
//
// Syntaxe :
//[ <Résultat> = ] GetRouteValue (<pTag> est chaîne)
//
// Paramètres:
// <ValueName> Nom de la valeur
// Valeur de retour :
// chaîne ANSI : // Valeur dans la route ou chaîne vide si inexistant
//
// Exemple pour la route /customer/{custid}/statistics
// CustomerID = oRequest:GetRouteValue("custid")
//
//
// Summary : Returns the value of a REST route variable
// Description of input/output parameters of 'GetRouteValue' :
//
// Syntax:
// GetRouteValue(ValueName est chaîne)
//
// Parameters :
// <ValueName> Value Name
//
// Return value:
// Value in the route or empty string if non-existent
//
// Example for route /customer/{custid}/statistics
// CustomerID = oRequest:GetRouteValue("custid")
FUNCTION PUBLIQUE GetRouteValue(LOCAL pTag est chaîne) : chaîne

RENVOYER :Vars[pTag]
```

Méthode GetURLValue

```
// Résumé : Renvoie une valeur d'URL de la requête REST
// Description des paramètres d'entrée/sortie de 'GetURLValue' :
//
// Syntaxe:
// GetURLValue(Header est chaîne)
//
// Paramètres :
```

```
// <valueName> Nom de la valeur
// Valeur de retour:
// Valeur dans l'URL ou chaîne vide si inexistant
//
// Exemple pour l'URL http://MonServeurLightREST.com:8080/client/123?unevaleur=ABC
// oRequest:GetURLValue("unevaleur") renverra ABC
//
//
//
// Summary : Returns a URL value of the REST request
// Description of input/output parameters of 'GetURLValue' :
//
// Syntaxe :
//[ <Résultat> = ] GetURLValue (<pTag> est chaîne)
//
// Parameters :
// <header> value name
// Valeur de retour / Return value:
// URL value or empty string if non-existent
//// Example for URL http://myLightRESTserver.com:8080/customer/123?anyvalue=ABC
// oRequest:GetHeaderValue("anyvalue") will return ABC
FONCTION PUBLIQUE GetURLValue(LOCAL pTag est chaîne) : chaîne

RENVOYER :URL.QueryValues[pTag]
```

Méthode VariantToSource

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
// VariantToSource (<pVariant>, <pSource> est objet source de données [, <pIDColumn> est chaîne])
//
// Paramètres :
// pVariant : <indiquez ici le rôle de pVariant>
// pSource (objet source de données) : <indiquez ici le rôle de pSource>
// pIDColumn (chaîne ANSI) : <indiquez ici le rôle de pIDColumn>
// Valeur de retour :
// Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCÉDURE VariantToSource(pVariant, pSource est Source de Données, LOCAL pIDColumn est chaîne = "")

taRubs          est tableau associatif (ccSansCasse) d'entiers
bTypeMemoPresent est booléen
sRub            est chaîne

(taRubs,bTypeMemoPresent) = ::__ListItem(pSource)

POUR TOUT sVal, sRub de pVariant
  SI PAS taRubs[sRub]..Existe _OU_ sRub ~= pIDColumn ALORS
    CONTINUER
  FIN

  SI PAS ::__IsItemMemo(taRubs[sRub]) ALORS
    {"pSource."+sRub, indVariable} = sVal
  FIN
FIN
```

IrServer

Code

Déclaration de IrServer

IrServer est Classe hérite IrDLL

PRIVÉ

```

    UUID                est chaîne

    sIPAndPort           est chaîne
    enHttpMode           est IrHTTPMode
    stSSLCertificate     est IrSSLCertificate
    stLetsEncrypt        est IrLetsEncrypt
    stSelfCertificate    est IrSelfCertificate
    stLogger             est IrLogger
    bMonitoring          est booléen
    enState              est IrServerState
    duRequestTimeout     est Durée
    eMaxRequests         est entier

    duReadTimeout        est Durée
    duWriteTimeout       est Durée
    duIdleTimeout        est Durée

    tstRoutes            est tableau associatif (ccSansCasse) de IrRoute

    vCustomData          est Variant

    fnCheckAuthentication est procédure

    fnHookEvent          est procédure

    taGlobalOptionsHeaders est tableau associatif (ccSansCasse+AvecDoublon) de chaîne
    taGlobalHeaders      est tableau associatif (ccSansCasse+AvecDoublon) de chaîne

    thThreadServer       est Thread
    thThreadRequest      est Thread

    bCipherringKey       est Buffer
    HashedCipherringKey  est Buffer

    eWinDevErrorDetails  est entier

    enHookedEvents       est IrEventLevel

```

FIN

```

// Déclaration de constantes spécifique
// Specific constant declaration
<BLOC const>
    CONSTANTE
        //Methods
        MethodGET      = "GET"
        MethodPOST     = "POST"
        MethodPUT       = "PUT"
        MethodDELETE    = "DELETE"
        MethodHEAD      = "HEAD"
        MethodPATCH    = "PATCH"

```

```

MethodOPTIONS      = "OPTIONS"

TAG_SHUTDOWN        = "///SHUTDOWN///"

DEFAULT_MAX_REQUEST  = 100
DEFAULT_REQUEST_TIMEOUT = 30s
DEFAULT_READ_TIMEOUT  = 30s
DEFAULT_WRITE_TIMEOUT = 30s
DEFAULT_IDLE_TIMEOUT  = 60s

SEMAPHORE_ROUTES      = "SEM_ROUTE_"
SEMAPHORE_HEADERS     = "SEM_HEADERS_"
SEMAPHORE_OPTION_HEADERS = "SEM_OPTION_HEADERS_"
SEMAPHORE_CUSTOM_SERVER = "SEM_CUSTOM_SERVER_"
FIN
<FIN>

// Déclaration de type spécifique nécessaire dans la classe
// Specific type declaration needed in the class
<BLOC types>
  IrServerState est Énumération
    STARTING
    STARTED
    STOPPING
    STOPPED
  FIN

// _stRoute est structure
//   route          est chaîne
//   method          est chaîne
//   procFonction est procédure
//   Timeout         est durée
// FIN

  IrSSLCertificate est Structure
    Certificate est chaîne
    PrivateKey  est chaîne
  FIN

  IrLogger est Structure
    bLogger          est booléen
    sLoggerFile      est chaîne
  FIN

  IrLetsEncrypt est Structure
    Domains          est tableau de chaînes
    Email            est chaîne
    CertificatePath  est chaîne
    RenewDelay       est entier
  FIN

  IrSelfCertificate est Structure
    KeySize est entier
  FIN

  IrEventLevel est Combinaison
    EVE_INFO
    EVE_REQUEST
    EVE_ERROR
  FIN

  IrEventType est Énumération
    EVT_INFO

    EVT_RECEIVED

```

```

    EVT_AUTH_FAILED
    EVT_BEFORE_METHOD
    EVT_AFTER_METHOD
    EVT_RESULT

    EVT_ERROR
    EVT_WARNING
    EVT_PANIC
FIN

lrEventInfo est Structure
    Level      est lrEventLevel
    Type       est lrEventType
    Request    est lrRequest
    Response   est lrResponse
    Message    est chaîne
FIN

lrHTTPMode est Énumération
    HTTP
    HTTPS_CERT
    HTTPS_SELF
    HTTPS_LETS_ENCRYPT
FIN
<FIN>

```

Méthode _copyAssociativeArray

```

//Cannot use WinDev ArrayCopy() because too strict with slightly different arrays
//Note : this method does not empty pDestArray before copying
PROCÉDURE PRIVÉE _copyAssociativeArray(pOrigArray, pDestArray)

POUR TOUT vVal, vTag de pOrigArray
    pDestArray[vTag] = vVal
FIN

```

Méthode _eventHook

```

FONCTION PRIVÉE _eventHook(LOCAL pLevel est lrEventLevel, LOCAL pType est lrEventType, pRequest est
lrRequest = Null, pResponse est lrResponse = Null, LOCAL pInfo = "") : booléen

stLoginfo      est lrEventInfo
bResult        est booléen

SI :fnHookEvent = Null OU PAS :enHookedEvents[pLevel] ALORS
    RENVOYER Vrai
FIN

stLoginfo.Level      = pLevel
stLoginfo.Type       = pType

SI pRequest <> Null ALORS
    stLoginfo.request = pRequest
FIN

SI pResponse <> Null ALORS
    stLoginfo.response = pResponse
FIN

stLoginfo.Message = pInfo

QUAND EXCEPTION DANS
    bResult = :fnHookEvent(stLoginfo)

```

```

SI pRequest <> Null _ET_ stLoginfo.request <> Null ALORS
    pRequest <= stLoginfo.Request
FIN
SI pResponse <> Null _ET_ stLoginfo.response <> Null ALORS
    pResponse <= stLoginfo.response
FIN

RENVOYER bResult

FAIRE
    //Hook function crashed, let's continue the calling process
    ExceptionActive()
    RENVOYER Vrai
FIN

```

Méthode _executeRequest

PROCÉDURE PRIVÉE _executeRequest(pRequest est Buffer)

```

///// :FIN /////
///// Exécution automatique en sortie de fonction /////

bOK          est booléen
sError       est chaîne

_stRESTRequest est Structure
    Body      est chaîne
    Method    est chaîne
    URL       est lrRequest.lrURL
    Proto     est chaîne
    // ProtoMajor est entier
    // ProtoMinor est entier
    Headers   est tableau associatif (ccSansCasse+AvecDoublon) de tableau de chaînes
    ContentLength est entier sur 8 octets
    // TransferEncoding est tableau de chaînes
    // Close          est booleen
    Host             est chaîne
    // Form           est tableau associatif (ccSansCasse) de chaînes
    // PostForm       est tableau associatif (ccSansCasse) de chaînes
    // MultipartForm  est lrform, serialise = faux
    Files            est tableau associatif (ccSansCasse) de lrRequest.lrFile
    // Trailer        est tableau associatif (ccSansCasse) de chaînes
    RemoteAddr       est chaîne
    RequestURI       est chaîne
    Authentication    est lrRequest.lrAuthentication
    // TLS:           r.TLS
    Vars              est tableau associatif (ccSansCasse) de chaînes
    CustomData        est chaîne, Sérialise = faux
FIN

_stRequest est Structure
    ID      est chaîne
    Route   est chaîne
    Method  est chaîne
    RestRequest est _stRESTRequest
FIN

_stResponse est Structure
    Timestamp est chaîne
    ID         est chaîne
    ContentType est chaîne
    Body       est chaîne
    Headers    est tableau associatif (ccSansCasse+AvecDoublon) de chaînes
    Status     est entier

```

```

    IsBinary est booléen
FIN

stRequest      est _stRequest
oRequest       est lrRequest
oResponse      est lrResponse
oRoute         est lrRoute
ChronoMeth     est Chrono
vEmpty         est Variant

QUAND EXCEPTION DANS
    Déserialise(stRequest, pRequest, psdJSON)
FAIRE
    :_panic("Error during internal deserialization " + CRLF + "JSON : " + pRequest + CRLF + CRLF +
ExceptionInfo(:eWinDevErrorDetails))
    oResponse.Status = lrResponse::StatusInternalServerError
    oResponse.Body   = "Erreur pendant la désérialisation interne (__executeRequest)"
ExceptionActive()
    RETOUR
FIN

oRequest      <= stRequest.RestRequest
oRequest:ID    = stRequest.ID

oRequest:CustomData = vEmpty

(bOK, oRoute) = :_getRoute(stRequest.Route + ESC + stRequest.Method)

SI PAS bOK ALORS
    //Impossible, but....
    :_panic("ERREUR : Route inconnue %1 %2", stRequest.Route, stRequest.Method)
    oResponse.ContentType = lrResponse::ContentTXT
    oResponse.Body        = ChaîneConstruit("ERREUR : Route inconnue %1 %2", stRequest.Route, st
Request.Method)
    oResponse.Status      = lrResponse::StatusNotFound
SINON

    oRequest.ServerCustomData = :CustomData
    oRequest.RouteCustomData = oRoute:CustomData

POUR TOUT header, name de stRequest.RestRequest.Headers
    POUR i = 1 _À header..Occurrence
        oRequest.Header[name] = header[i]
    FIN
FIN

QUAND EXCEPTION DANS

    SI PAS :_eventHook(EVE_REQUEST, EVT_RECEIVED, oRequest, oResponse) ALORS
        RETOUR
    FIN

    SI :fnCheckAuthentication <> Null _ET_ PAS :fnCheckAuthentication(oRequest) _ET_      //
Authentication function fails
        PAS :_eventHook(EVE_REQUEST, EVT_AUTH_FAILED, oRequest, oResponse) ALORS      //
Authentication HOOK fails

        SI oResponse.Status=0 ALORS
            //No status was set, default = StatusUnauthorized
            oResponse.Status = lrResponse::StatusUnauthorized
        FIN

        RETOUR

    FIN

```

```

SI oRoute.ProcFonction = Null _ET_ stRequest.Method = IrServer::MethodOPTIONS ALORS
    SémaphoreDébut(::SEMAPHORE_HEADERS+:UUID)
    SI :taGlobalOptionsHeaders..Occurrence>0 ALORS
        oRoute.ProcFonction = :_returnGlobalOptionsHeaders
    FIN
    SémaphoreFin(::SEMAPHORE_HEADERS+:UUID)
FIN

SI :bMonitoring ALORS
    ChronoDébut(ChronoMeth)
FIN

SI PAS :_eventHook(EVE_REQUEST, EVT_BEFORE_METHOD, oRequest, oResponse) ALORS
    RETOUR
FIN

oResponse = oRoute.ProcFonction(oRequest)

SI PAS :_eventHook(EVE_REQUEST, EVT_AFTER_METHOD, oRequest, oResponse) ALORS
    RETOUR
FIN

FAIRE
    :_eventHook(EVE_ERROR, EVT_ERROR, oRequest, *, ExceptionInfo(:eWinDevErrorDetails))
    oResponse.Body = ExceptionInfo(:eWinDevErrorDetails)
    oResponse.Status = IrResponse::StatusInternalServerError
    ExceptionActive()
    RETOUR
FIN
FIN

////////////////////////////////////
//The following code is always executed before function returns
////////////////////////////////////
FIN:

SI oResponse.Status = 0 ALORS
    :_eventHook(EVE_ERROR, EVT_WARNING, oRequest, oResponse,
        "Le statut de la réponse, qui était égal à 0, a été affecté à 200 OK")
    oResponse.Status = IrResponse::StatusOK
FIN

SI :bMonitoring ALORS
    ChronoFin(ChronoMeth)
    oResponse.Headers["Chrono-Methode-Millisecon"] = ChronoMeth..Valeur..EnMillisecondes
FIN

SI :stLogger.bLogger ALORS
    oResponse.Headers["Warning-LightREST-Logger"] =
        "***WARNING*** LightREST logger is enabled, global performance may be degraded"
FIN
SémaphoreDébut(::SEMAPHORE_HEADERS+:UUID)
:_copyAssociativeArray(:taGlobalHeaders, oResponse.Headers)
SémaphoreFin(::SEMAPHORE_HEADERS+:UUID)

:_eventHook(EVE_REQUEST, EVT_RESULT, oRequest, oResponse)

// stResponse      <= oResponse
// stResponse.ID    = stRequest.ID
//
// Sérialise(stResponse, bResponse, psdJSON)

(bOK, sError) = :apiSetRequestResponse(oRequest.ID, oResponse)
SI PAS bOK ALORS

```

```

        :_panic(oRequest:ID + " : Cannot set request response : "+sError)
    FIN

```

Méthode _getGolangBuffer

```

//
FONCTION PRIVÉE _getGolangBuffer(pPointer est entier système) : Buffer

bRes est Buffer

bRes = ChaîneRécupère(pPointer, crAdresseASCII)
//:apiFreeCString(pPointer)

RENVOYER bRes

```

Méthode _getRoute

```

FONCTION PRIVÉE _getRoute(LOCAL pRoute est chaîne) : (booléen, lrRoute)

oRoute est lrRoute
bOK est booléen

SémaphoreDébut(::SEMAPHORE_ROUTES+:UUID)
    bOK = :tstRoutes[pRoute]..existe
    SI bOK ALORS
        oRoute = :tstRoutes[pRoute]
    FIN
SémaphoreFin(::SEMAPHORE_ROUTES+:UUID)

RENVOYER (bOK, oRoute)

```

Méthode _getWinDevErrorDetails

```

FONCTION PRIVÉE _getWinDevErrorDetails() : entier

RENVOYER :eWinDevErrorDetails

```

Méthode _panic

```

PROCÉDURE PRIVÉE _panic(*) : booléen

:_eventHook(EVE_ERROR, EVT_PANIC, *, *, ChaîneConstruit(MesParamètres))

RENVOYER fAjouteTexte(ComplèteRep(fRepExe())+"PANIC_LR_"+DateSys()+HeureSys()+".txt", ChaîneConstruit(
MesParamètres)+CRLF)

```

Méthode _returnGlobalOptionsHeaders

```

FONCTION PRIVÉE _returnGlobalOptionsHeaders(pRequest est lrRequest <utile>) : lrResponse

oResponse est lrResponse

SémaphoreDébut(::SEMAPHORE_OPTION_HEADERS+:UUID)
    :_copyAssociativeArray(:taGlobalOptionsHeaders, oResponse:Headers)
SémaphoreFin(::SEMAPHORE_OPTION_HEADERS+:UUID)

oResponse.Status = lrResponse::StatusNoContent

RENVOYER oResponse

```

Méthode AddRoute

```
//Kept for V2 compatibility
FONCTION AddRoute(LOCAL pRoute est chaîne, LOCAL pMethod est chaîne, pFunction est procédure(lrRequest)
, LOCAL pTimeout est Durée = :duRequestTimeout) : (booléen, chaîne)

oRoute est lrRoute
bOK est booléen
sError est chaîne

oRoute.Route          = pRoute
oRoute.Method          = pMethod
oRoute.ProcFonction    = pFunction
oRoute.Timeout         = pTimeout

(bOK, sError) = :AddRoute(oRoute)
RENOYER (bOK, sError)
```

Méthode AddRoute

```
PROCÉDURE AddRoute(pRoute est lrRoute) : (booléen, chaîne)
```

```
oOptionsRoute est lrRoute
```

```
//This RegEx should be fine. It works on any check tool I tested. But sometimes not with WinDev, ie does not
work with "/image". Don't know why :-{
```

```
//SI pas pRoute.NoRegex _et_ PAS VérifieExpressionRégulière(pRoute:route,
"^/(?:$|(?:[A-Za-z0-9_-]+|\\{[A-Za-z0-9_-]+\\})?(?:/(?:[A-Za-z0-9_-]+|\\{[A-Za-z0-9_-]+\\}))*$)") ALORS
```

```
// RENOYER (Faux, "Syntaxe de la route incorrecte")
//FIN
```

```
:_eventHook(EVE_INFO, EVT_INFO, *, *, ChaîneConstruit("Ajout de la route %1 méthode %2", pRoute:route,
pRoute:method))
```

```
SémaphoreDébut(::SEMAPHORE_ROUTES+:UUID)
:tstRoutes[pRoute.route + ESC + pRoute.method] = pRoute
```

```
SI pRoute.UseGlobalOptionHeaders ALORS
  //oOptionsRoute <= pRoute
  oOptionsRoute.Route = pRoute.route
  oOptionsRoute.Method = ::MethodOPTIONS
  oOptionsRoute.ProcFonction = Null
  :tstRoutes[oOptionsRoute.Route + ESC + oOptionsRoute.Method] = oOptionsRoute
```

```
FIN
SémaphoreFin(::SEMAPHORE_ROUTES+:UUID)
```

```
RENOYER (Vrai, "")
```

Méthode CipherID

```
FONCTION CipherID(LOCAL pIDName est chaîne, LOCAL pIDValue est chaîne) : chaîne
```

```
bOK          est booléen
sError       est chaîne
ePtr         est entier système
```

```
(bOK, ePtr, sError) = :apiCipherID(pIDName, pIDValue, :HashedCipheringKey)
```

```
SI PAS bOK ALORS
  RENOYER ""
```

```
FIN
RENOYER :_getGolangBuffer(ePtr)
```

Méthode CipherIDs

```

FONCTION CipherIDs(LOCAL pIDName est chaîne, pIDValues) : tableau de chaînes

sIDs          est chaîne
sSeparator     est chaîne = ESC
tCipheredIDs  est tableau de chaînes
bOK            est booléen
sError         est chaîne
ePtr           est entier système

sIDs = TableauVersChaîne(pIDValues,sSeparator)

(bOK, ePtr, sError) = :apiCipherIDs(pIDName, sIDs, sSeparator, :HashedCipherringKey)
SI bOK ALORS
    ChaîneVersTableau(:_getGolangBuffer(ePtr), tCipheredIDs, sSeparator)
FIN
RENOYER tCipheredIDs

```

Constructeur

```

PROCÉDURE Constructeur()

vEmpty est Variant

:UUID = DonneUUID()

SémaphoreCrée(::SEMAPHORE_CUSTOM_SERVER+:UUID, partageAucun)
SémaphoreCrée(::SEMAPHORE_ROUTES+:UUID, partageAucun)
SémaphoreCrée(::SEMAPHORE_HEADERS+:UUID, partageAucun)
SémaphoreCrée(::SEMAPHORE_OPTION_HEADERS+:UUID, partageAucun)

:enState          = STOPPED
:eWinDevErrorDetails = errCompleet
:CipherringKey     = :UUID
:duRequestTimeout  = ::DEFAULT_REQUEST_TIMEOUT
:enHttpMode        = HTTP
:CustomData        = vEmpty
:eMaxRequests      = ::DEFAULT_MAX_REQUEST
:duReadTimeout     = ::DEFAULT_READ_TIMEOUT
:duWriteTimeout    = ::DEFAULT_WRITE_TIMEOUT
:duIdleTimeout     = ::DEFAULT_IDLE_TIMEOUT

```

Méthode DecipherID

```

FONCTION DecipherID(LOCAL pIDName est chaîne, LOCAL pIDValue est chaîne) : chaîne

bOK          est booléen
sError       est chaîne
ePtr         est entier système

(bOK, ePtr, sError) = :apiDecipherID(pIDName, pIDValue, :HashedCipherringKey)
SI PAS bOK ALORS
    RENOYER ""
FIN
RENOYER :_getGolangBuffer(ePtr)

```

Méthode DecipherIDs

```

FONCTION DecipherIDs(LOCAL pIDName est chaîne, pIDValues) : tableau de chaînes

sSeparator     est chaîne = ESC
tDecipheredIDs est tableau de chaînes
bOK            est booléen
sError         est chaîne

```

```

ePtr          est entier système
sIDs          est chaîne

sIDs = TableauVersChaîne(pIDValues,sSeparator)

(bOK, ePtr, sError) = :apiDecipherIDs(pIDName, sIDs, sSeparator, :HashedCipherringKey)
SI bOK ALORS
    ChaîneVersTableau(:_getGolangBuffer(ePtr), tDecipheredIDs, sSeparator)
FIN
RENOYER tDecipheredIDs

```

Destructeur

```

PROCÉDURE Destructeur()

SI :enState = STARTED _OU_ :enState = STARTING ALORS
    //Properly terminate current processes
    :Terminate()

    //Kill remaining processes
    :Kill()
FIN

SI :thThreadRequest..Etat <> threadInexistent ALORS
    ThreadArrête(:thThreadRequest..Nom)
FIN

SI :thThreadServer..Etat <> threadInexistent ALORS
    ThreadArrête(:thThreadServer..Nom)
FIN

SémaphoreDétruit(::SEMAPHORE_CUSTOM_SERVER+:UUID)
SémaphoreDétruit(::SEMAPHORE_ROUTES+:UUID)
SémaphoreDétruit(::SEMAPHORE_HEADERS+:UUID)
SémaphoreDétruit(::SEMAPHORE_OPTION_HEADERS+:UUID)

```

Récupération de la propriété CipherringKey

```

FONCTION PUBLIQUE CipherringKey() : Buffer

RENOYER :bCipherringKey

```

Affectation de la propriété CipherringKey

```

PROCÉDURE PUBLIQUE CipherringKey( LOCAL pKey est Buffer)

:bCipherringKey = pKey
:HashedCipherringKey = Encode(HashChaîne(HA_SHA_256, :bCipherringKey), encodeBASE64SansRC)

```

Récupération de la propriété CustomData

```

PROCÉDURE PUBLIQUE CustomData()

vCustomData est Variant

SémaphoreDébut(::SEMAPHORE_CUSTOM_SERVER+:UUID)
vCustomData = :vCustomData
SémaphoreFin(::SEMAPHORE_CUSTOM_SERVER+:UUID)

RENOYER vCustomData

```

Affectation de la propriété CustomData

```
PROCÉDURE PUBLIQUE CustomData(pCustomData)
```

```
SémaphoreDébut(::SEMAPHORE_CUSTOM_SERVER+:UUID)
:vCustomData = pCustomData
SémaphoreFin(::SEMAPHORE_CUSTOM_SERVER+:UUID)
```

Récupération de la propriété HTTPMode

```
FONCTION PUBLIQUE HTTPMode() : lrHTTPMode
```

```
REVOYER :enHTTPMode
```

Affectation de la propriété HTTPMode

```
PROCÉDURE PUBLIQUE HTTPMode(LOCAL pHTTPMode est lrHTTPMode)
```

```
:enHTTPMode = pHTTPMode
```

Récupération de la propriété IdleTimeout

```
PROCÉDURE PUBLIQUE IdleTimeout() : Durée
```

```
REVOYER :duIdleTimeout
```

Affectation de la propriété IdleTimeout

```
PROCÉDURE PUBLIQUE IdleTimeout(pTimeout est une Durée)
```

```
:duIdleTimeout=pTimeout
```

Récupération de la propriété IPAndPort

```
FONCTION PUBLIQUE IPAndPort() : chaîne
```

```
REVOYER :sIPAndPort
```

Affectation de la propriété IPAndPort

```
PROCÉDURE PUBLIQUE IPAndPort(LOCAL pIPAndPort est chaîne)
```

```
:sIPAndPort = pIPAndPort
```

Récupération de la propriété Logger

```
//Logger is just for debugging LightREST. Do not activate it in a production environment.
```

```
FONCTION PUBLIQUE Logger() : booléen
```

```
REVOYER :stLogger:bLogger
```

Affectation de la propriété Logger

```
//Logger is just for debugging LightREST. Do not activate it in a production environment.
```

```
PROCÉDURE PUBLIQUE Logger( LOCAL pLogger est booléen)
```

```
SI pLogger <> :stLogger:bLogger ALORS
  SI pLogger ALORS
    :SetLoggerOn(:stLogger:sLoggerFile)
  SINON
    :SetLoggerOff()
  FIN
  :stLogger:bLogger = pLogger
```

FIN

Récupération de la propriété LoggerFileFONCTION PUBLIQUE `LoggerFile()` : chaîneRENNVOYER : `stLogger.sLoggerFile`**Affectation de la propriété LoggerFile**PROCÉDURE PUBLIQUE `LoggerFile( LOCAL pFile est chaîne)`SI `pFile <> :stLogger.sLoggerFile _ET_ :stLogger.bLogger` ALORS
: `SetLoggerOn(pFile)`

SINON

: `stLogger.sLoggerFile=pFile`

FIN

Récupération de la propriété MaxRequestsPROCÉDURE PUBLIQUE `MaxRequests()`RENNVOYER : `eMaxRequests`**Affectation de la propriété MaxRequests**PROCÉDURE PUBLIQUE `MaxRequests(pMaxRequests est entier)`: `eMaxRequests = pMaxRequests`**Récupération de la propriété Monitoring**FONCTION PUBLIQUE `Monitoring()` : booléenRENNVOYER : `bMonitoring`**Affectation de la propriété Monitoring**PROCÉDURE PUBLIQUE `Monitoring( LOCAL pMonitoring est booléen)`: `bMonitoring = pMonitoring`**Récupération de la propriété ReadTimeout**PROCÉDURE PUBLIQUE `ReadTimeout()`RENNVOYER : `duReadTimeout`**Affectation de la propriété ReadTimeout**PROCÉDURE PUBLIQUE `ReadTimeout(pTimeout est Durée)`: `duReadTimeout = pTimeout`**Récupération de la propriété RequestTimeout**FONCTION PUBLIQUE `RequestTimeout()` : DuréeRENNVOYER : `duRequestTimeout`**Affectation de la propriété RequestTimeout**

PROCÉDURE PUBLIQUE RequestTimeout(LOCAL pTimeout est Durée)

:duRequestTimeout = pTimeout

Récupération de la propriété State

FONCTION PUBLIQUE State() : IrServerState

RENOYER :enState

Récupération de la propriété TempFolder

FONCTION PUBLIQUE TempFolder() : chaîne

RENOYER :cTempFolder

Affectation de la propriété TempFolder

PROCÉDURE PUBLIQUE TempFolder(LOCAL pFolder est chaîne)

:cTempFolder = ComplèteRep(pFolder)

Récupération de la propriété WindevErrorDetails

FONCTION PUBLIQUE WindevErrorDetails() : entier

RENOYER :eWinDevErrorDetails

Affectation de la propriété WindevErrorDetails

PROCÉDURE PUBLIQUE WindevErrorDetails(LOCAL pDetails est entier)

:eWinDevErrorDetails = pDetails

Récupération de la propriété WriteTimeout

PROCÉDURE PUBLIQUE WriteTimeout() : Durée

RENOYER :duWriteTimeout

Affectation de la propriété WriteTimeout

PROCÉDURE PUBLIQUE WriteTimeout(pTimeout est une Durée)

:duWriteTimeout=pTimeout

Méthode Kill

FONCTION Kill() : (booléen, chaîne)

bOK est booléen

sError est chaîne

SI :enState <> STARTED ALORS

RENOYER (Faux, "Le serveur n'est pas démarré")

FIN

:enState = STOPPING

:_eventHook(EVE_INFO, EVT_INFO, *, *, ChaîneConstruit("Serveur à l'écoute de %1 en cours de meurtre", :IPAndPort))

(bOK, sError)= :apiKillServer()

```

SI PAS bOK ALORS
    RENVOYER (Faux, sError)
FIN

SI sError <> "" ALORS
    RENVOYER (Faux, UTF8VersChaîne(sError))
FIN

:_eventHook(EVE_INFO, EVT_INFO, *, *, ChaîneConstruit("Serveur à l'écoute de %1 tué", :IPAndPort))

:enState = STOPPED

RENOYER (Vrai, "")

```

Méthode SetAuthenticationCheckFunction

```

PROCÉDURE SetAuthenticationCheckFunction(pFunction est procédure)

:fnCheckAuthentication = pFunction

```

Méthode SetEventHookFunction

```

PROCÉDURE SetEventHookFunction(pFunction est procédure, LOCAL pEvents est lrEventLevel)

:fnHookEvent      = pFunction
:enHookedEvents = pEvents

```

Méthode SetGlobalHeader

```

PROCÉDURE SetGlobalHeader(pHeader est chaîne, pValue est chaîne)

SémaphoreDébut(::SEMAPHORE_HEADERS+:UUID)
    :taGlobalHeaders[pHeader] = pValue
SémaphoreFin(::SEMAPHORE_HEADERS+:UUID)

```

Méthode SetGlobalHeaders

```

PROCÉDURE SetGlobalHeaders(LOCAL pGlobalHeaders est tableau associatif de chaînes)

SémaphoreDébut(::SEMAPHORE_HEADERS+:UUID)
    :_copyAssociativeArray(pGlobalHeaders, :taGlobalHeaders)
SémaphoreFin(::SEMAPHORE_HEADERS+:UUID)

```

Méthode SetGlobalOptionsHeader

```

PROCÉDURE SetGlobalOptionsHeader(pHeader est chaîne, pValue est chaîne)

SémaphoreDébut(::SEMAPHORE_OPTION_HEADERS+:UUID)
    :taGlobalOptionsHeaders[pHeader] = pValue
SémaphoreFin(::SEMAPHORE_OPTION_HEADERS+:UUID)

```

Méthode SetGlobalOptionsHeaders

```

PROCÉDURE SetGlobalOptionsHeaders(LOCAL pGlobalOptionsHeaders est tableau associatif de chaînes)

SémaphoreDébut(::SEMAPHORE_OPTION_HEADERS+:UUID)
    :_copyAssociativeArray(pGlobalOptionsHeaders, :taGlobalOptionsHeaders)
SémaphoreFin(::SEMAPHORE_OPTION_HEADERS+:UUID)

```

Méthode SetHTTPTSCertificate

PROCÉDURE SetHTTSPCertificate(pCertificate est chaîne, pPrivateKey est chaîne)

```
:stSSLCertificate.Certificate = pCertificate
:stSSLCertificate.PrivateKey = pPrivateKey
```

Méthode SetHTTSPCertificate

PROCÉDURE SetHTTSPCertificate(pCertificate est IrSSLCertificate)

```
:stSSLCertificate = pCertificate
```

Méthode SetLetsEncryptParameters

PROCÉDURE SetLetsEncryptParameters(LOCAL pDomain est chaîne, LOCAL pEmail est chaîne, LOCAL pCertificatePath est chaîne = "", LOCAL pRenewDelay est entier = 0)

tDomains est tableau de chaîne

```
SI pDomain <> "" ALORS
    tDomains.Ajoute(pDomain)
FIN
```

```
:stLetsEncrypt.Domains      = tDomains
:stLetsEncrypt.Email        = pEmail
:stLetsEncrypt.CertificatePath = pCertificatePath
:stLetsEncrypt.RenewDelay    = pRenewDelay
```

Méthode SetLetsEncryptParameters

PROCÉDURE SetLetsEncryptParameters(LOCAL pDomains est tableau de chaînes, LOCAL pEmail est chaîne, LOCAL pCertificatePath est chaîne = "", LOCAL pRenewDelay est entier = 0)

```
:stLetsEncrypt.Domains      = pDomains
:stLetsEncrypt.Email        = pEmail
:stLetsEncrypt.CertificatePath = pCertificatePath
:stLetsEncrypt.RenewDelay    = pRenewDelay
```

Méthode SetLetsEncryptParameters

PROCÉDURE SetLetsEncryptParameters(LOCAL pLetsEncrypt est IrLetsEncrypt)

```
:stLetsEncrypt = pLetsEncrypt
```

Méthode SetLoggerOff

//Logger is just for debugging LightREST. Do not activate it in a production environment.
FONCTION SetLoggerOff() : (booléen, chaîne)

```
sError      est chaîne
bOK          est booléen
ePtr         est entier système
```

```
:stLogger.bLogger = False
(bOK, ePtr, sError) = :apiSetLoggerOff()
```

RENVoyer (bOK, sError)

Méthode SetLoggerOn

//Logger is just for debugging LightREST. Do not activate it in a production environment.

FONCTION SetLoggerOn(pLoggerFile est chaîne) : (booléen, chaîne)

```

bOK          est booléen
sError       est chaîne
ePtr         est entier système

SI pLoggerFile = "" ALORS
  pLoggerFile = ComplèteRep(fRepExe()+"lr_log.txt"
FIN

:stLogger.bLogger = True
:stLogger.sLoggerFile = pLoggerFile

(bOK, ePtr, sError) = :apiSetLoggerOn(ChaîneVersUTF8(pLoggerFile))

RENOYER (bOK, sError)

```

Méthode SetSelfCertificateParameters

```

PROCÉDURE SetSelfCertificateParameters(plrSelfCertificate est lrSelfCertificate)

:stSelfCertificate = plrSelfCertificate

```

Méthode Start

```

FONCTION Start() : (booléen, chaîne)

serverData est JSON
sStartError est chaîne
bOK est booléen
sError est chaîne

SI :IPAndPort = "" ALORS
  sStartError = "L'interface réseau et le port d'écoute ne sont pas été définis ! "
  :_eventHook(EVE_ERROR, EVT_ERROR, *, *, sStartError)
  RENVOYER (Faux, sStartError)
FIN

SI :enState <> STOPPED ALORS
  sStartError = "Cette instance du serveur est déjà lancée"
  :_eventHook(EVE_ERROR, EVT_ERROR, *, *, sStartError)
  RENVOYER (Faux, sStartError)
FIN

serverData.IPAndPort      = :sIPAndPort
serverData.Logger         = :stLogger.bLogger
serverData.LoggerPath     = :stLogger.sLoggerFile
serverData.Monitoring     = :bMonitoring
serverData.HttpMode       = :enHttpMode..Nom
serverData.MaxRequests    = :eMaxRequests
serverData.Timeouts.ReadTimeout = :duReadTimeout..EnSecondes
serverData.Timeouts.WriteTimeout = :duWriteTimeout..EnSecondes
serverData.Timeouts.IdleTimeout = :duIdleTimeout..EnSecondes

//Stting routes
POUR TOUT ELEMENT oRoute DE :tstRoutes
  (bOK, sError) = :apiAddRoute(oRoute)
  SI PAS bOK ALORS
    :_eventHook(EVE_ERROR, EVT_ERROR, *, *, ChaîneConstruit(
      "Erreur lors de l'ajout de la route %1 méthode %2 : %3", oRoute.Route, oRoute.Method, sError))
    RENVOYER (Faux, sError)
  FIN
FIN

SELON :enHttpMode
  CAS HTTP

```

```

CAS HTTPS_SELF
    serverData.KeySize = :stSelfCertificate.KeySize

CAS HTTPS_CERT
    SI :stSSLCertificate:Certificate = "" _OU_ :stSSLCertificate.PrivateKey = "" ALORS
        sStartError = "En mode HTTPS mais aucun certificat n'a été défini"
        :_eventHook(EVE_ERROR, EVT_ERROR, *, *, sStartError)
        RENVOYER (Faux, sStartError)
    FIN

    serverData.Certificate.Certificate = :stSSLCertificate.Certificate
    serverData.Certificate.PrivateKey = :stSSLCertificate.PrivateKey

CAS HTTPS_LETS_ENCRYPT
    SI :stLetsEncrypt:Domains..Occurrence = 0 _OU_ :stLetsEncrypt:Email = "" ALORS
        sStartError = "En mode HTTPS LET'S ENCRYPT, mais aucun domaine et/ou aucun email
            n'ont été définis"
        :_eventHook(EVE_ERROR, EVT_ERROR, *, *, sStartError)
        RENVOYER (Faux, sStartError)
    FIN

    serverData.LetsEncrypt.Email = :stLetsEncrypt.Email
    serverData.LetsEncrypt.CertificatePath = :stLetsEncrypt.CertificatePath
    serverData.LetsEncrypt.Domains = :stLetsEncrypt.Domains
    serverData.LetsEncrypt.RenewDelay = :stLetsEncrypt.RenewDelay
FIN

:enState = STARTING

:thThreadServer = ThreadExécute(piStartServer, (), threadCopieComplèteContexteHFSQL+
threadAttendDémarriage+threadContexteGlobal)

//Wait for server to start all processes
ThreadPause(1s)

SI sStartError <> "" ALORS
    :_eventHook(EVE_ERROR, EVT_ERROR, *, *, sStartError)
    :enState = STOPPED
    RENVOYER (Faux, sStartError)
FIN

//Thread Waiting for REST requests
:thThreadRequest = ThreadExécute(piGetRequest, (), threadCopieComplèteContexteHFSQL+
threadAttendDémarriage+threadContexteGlobal)

:enState = STARTED

:_eventHook(EVE_INFO, EVT_INFO, *, *, "Serveur LightREST démarré sur "+IPAndPort)

RENVOYER (Vrai, "")

PROCÉDURE INTERNE piStartServer()
    (bOK, sStartError) = :apiStartServer(serverData..FormatJSON)
FIN

PROCÉDURE INTERNE piGetRequest()
    bRequest est Buffer

    BOUCLE
        bRequest = :apiWaitForRequest()
        SI bRequest <> "" ALORS
            SI bRequest = ::TAG_SHUTDOWN ALORS

```

```

        //Server shutdown, exit loop
        SORTIR
    FIN
    ThreadExécute(:_executeRequest, (bRequest), threadCopieComplèteContexteHFSQL+
threadContexteGlobal)
FIN
    SI ThreadArrêtDemandé() ALORS
        SORTIR
    FIN
FIN
FIN

```

Méthode Terminate

PROCÉDURE `Terminate`(LOCAL `pTimeout` est `Durée` = ::DEFAULT_REQUEST_TIMEOUT) : (booléen, chaîne)

`bOK` est booléen
`sError` est chaîne

```

SI :enState <> STARTED ALORS
    RENVOYER (Faux, "Le serveur n'est pas démarré")
FIN

```

```

:_eventHook(EVE_INFO, EVT_INFO, *, *, ChaîneConstruit("Serveur à l'écoute de %1 en cours d'arrêt", :
IPAndPort))
:enState = STOPPING
:_eventHook(EVE_INFO, EVT_INFO, *, *, ChaîneConstruit("En attente de la fin des traitements en cours",
:IPAndPort))

```

```

(bOK, sError) = :apiStopServer(pTimeout)
SI PAS bOK ALORS
    RENVOYER (Faux, sError)
SINON SI sError<>"" ALORS
    RENVOYER (Faux, UTF8VersChaîne(sError))
FIN

```

```

:enState = STOPPED

```

```

:_eventHook(EVE_INFO, EVT_INFO, *, *, ChaîneConstruit("Serveur à l'écoute de %1 arrêté", :IPAndPort))

```

```

RENOYER (Vrai, "")

```

IrSession

Code

Déclaration de IrSession

```

IrSession est une Classe
  PRIVÉ
    cID                est chaîne
    eLastErrorCode     est entier

  PRIVÉ GLOBAL
    //Tous les thread qui utilisent les sessions doivent partager ces membres pour qu'un thread puisse
    //déchiffrer les données chiffrées par un autre

    //All threads using sessions must share these members so that a thread can decipher data ciphered
    //by another one

    cSessionsFolder    est chaîne
    cSessionSalt        est chaîne
    eSignatureSize      est entier
    bCipherringKey      est Buffer
    bKeyHash            est Buffer
    bCheckSignature     est booléen
    thThreadCleanup     est Thread
FIN

<BLOC const>
  CONSTANTE

    SEMAPHORE_SESSIONS = "SEM_SESSIONS_"

    ErrorSessionExpired      = 0x1000
    ErrorSessionIllegal      = 0x1001
    ErrorSessionCannotLoad   = 0x1002
    ErrorSessionCannotSave   = 0x1003
    ErrorSessionCannotSetValue = 0x1004
    ErrorSessionCannotDestroy = 0x1005
    ErrorSessionCannotDecipher = 0x1006
    ErrorSessionCannotGetValue = 0x1007
  FIN
<FIN>

```

Méthode __sessionCheckSignature

```

FONCTION PRIVÉE __sessionCheckSignature() : booléen

  cSessionID      est chaîne
  cSignature       est chaîne
  cSignatureCheck  est chaîne

  SI PAS ::bCheckSignature ALORS
    RENVOYER Vrai
  FIN

  cSignature      = Droite(:ID, ::eSignatureSize)
  cSessionID      = :ID[[1 À Taille(:ID)-::eSignatureSize]]

  cSignatureCheck  = Minuscule(BufferVersHexa(HashChaîne(HA_MURMUR_1, cSessionID, ::cSessionSalt),

```

SansRegroupement))

RENOYER (cSignature = cSignatureCheck)

Méthode __sessionRead

FONCTION PRIVÉE __sessionRead() : (booléen, entier, chaîne)

bSessionData est Buffer

bSessionData = fChargeBuffer(::cSessionsFolder+.ID+".session")

SI ErreurDétectée() ALORS

RENOYER (Faux, ::ErrorSessionCannotLoad, "")

FIN

SI ::bKeyHash <> "" ALORS

bSessionData = DécrypteStandard(bSessionData, ::bKeyHash, crypteAES256)

SI ErreurDétectée() ALORS

RENOYER (Faux, ::ErrorSessionCannotDecipher, "")

FIN

FIN

RENOYER (Vrai, 0, bSessionData)

Méthode __sessionsCleanup

PROCÉDURE GLOBALE PRIVÉE __sessionsCleanup()

cSessionFiles est chaîne

cSessionID est chaîne

cSemaphore est chaîne

bOK est booléen

eErr est entier

jsSession est JSON

vSessionData est Variant

dhLastAccess est DateHeure

dhNow est une DateHeure

oSession est IrSession

cSessionFiles = fListeFichier(::cSessionsFolder+"*.session")

POUR TOUTE CHAÎNE cFile DE cSessionFiles SÉPARÉE PAR CRLF

cSessionID = fExtraitChemin(cFile, fFichier)

cSemaphore = ::SEMAPHORE_SESSIONS+cSessionID

SémaphoreDébut(cSemaphore)

//Okazou la session a été tuée entre temps...

//In case if the session was ended in meantime

SI fFichierExiste(cFile) ALORS

oSession:ID = cSessionID

(bOK, eErr, jsSession) = oSession:__sessionRead()

SI bOK ALORS

QUAND EXCEPTION DANS

vSessionData = JSONVersVariant(jsSession)

dhLastAccess = vSessionData._session.TsLastAccess

dhLastAccess.Seconde += Val(vSessionData._session.timeout)

dhNow = DateSys()+HeureSys()

SI dhLastAccess < dhNow ALORS

//Session expired

SémaphoreFin(cSemaphore)

oSession:Destroy()

CONTINUER

FIN

```

        FAIRE
            ExceptionActive()
            //File with illegal content, destroy session file
            fSupprime(cFile)
        FIN
    FIN
    SémaphoreFin(cSemaphore)
FIN

```

Méthode __sessionWrite

```

FONCTION PRIVÉE __sessionWrite(LOCAL jsSessionData est JSON) : (booléen, entier, chaîne)

bSessionData est Buffer = JSONVersChaîne(jsSessionData)

SI ::bCipherringKey<>"" ALORS
    bSessionData = CrypteStandard(bSessionData, ::bKeyHash, crypteAES256)
FIN

SI PAS fSauveBuffer(::cSessionsFolder+:ID+".session", bSessionData) _OU_ ErreurDétectée() ALORS
    RENVOYER (Faux, ::ErrorSessionCannotSave, "")
FIN

RENOYER (Vrai, 0, bSessionData)

```

Méthode __TimeStamp

```

FONCTION PRIVÉE GLOBALE __TimeStamp() : chaîne

RENOYER DateSys()+HeureSys()[[1 À 8]]

```

Méthode Check

```

FONCTION Check() : (booléen, entier)

SI PAS :__sessionCheckSignature() ALORS
    :eLastErrorCode = ::ErrorSessionIllegal
    RENVOYER (Faux, :eLastErrorCode)
FIN

SI PAS fFichierExiste(::cSessionsFolder+:ID+".session") ALORS
    :eLastErrorCode = ::ErrorSessionExpired
    RENVOYER (Faux, :eLastErrorCode)
FIN

RENOYER (Vrai, 0)

```

Méthode Cleanup

```

PROCÉDURE Cleanup()

::__sessionsCleanup()

```

Constructeur

```

PROCÉDURE Constructeur()

SI ::cSessionsFolder = "" ALORS
    ::cSessionsFolder = ComplèteRep(ComplèteRep(fRepExe()) + "$LR_SESSIONS")
FIN

```

```

SI ::eSignatureSize = 0 ALORS
  ::cSessionSalt = "$!r$SALT!"
  ::eSignatureSize = Taille(BufferVersHexa(HashChaîne(HA_MURMUR_1, "X", ::cSessionSalt),
    SansRegroupement))
FIN

```

Méthode Create

```

// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'Create' :
//
// Syntaxe :
//[ <Résultat> = ] Create ( [<pTimeout> est durée])
//
// Paramètres :
// pTimeout (durée - valeur par défaut=0010000000) : <indiquez ici le rôle de pTimeout>
// Valeur de retour :
// booléen : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
// Détaillez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION Create(LOCAL pTimeout est Durée = 3600s) : booléen

cSessionID    est chaîne
jsSession     est JSON
cSignature     est chaîne
bOK           est booléen
eErr          est entier
sResVal       est chaîne

SI PAS fRepCrée(::cSessionsFolder) ALORS
  RENVOYER Faux
FIN

cSessionID      = DonneGUID(guidBrut256)
cSignature       = Minuscule(BufferVersHexa(HashChaîne(HA_MURMUR_1, cSessionID, ::
cSessionSalt),SansRegroupement))
cSessionID      += cSignature
:ID             = cSessionID
jsSession.ID     = :ID
jsSession._session.TsCreated = ::__TimeStamp()
jsSession._session.TsLastAccess = jsSession._session.TsCreated
jsSession._session.Timeout = pTimeout..EnSecondes

(bOK, eErr, sResVal) = :__sessionWrite(jsSession)

SémaphoreCrée(::SEMAPHORE_SESSIONS+cSessionID, partageAucun)

SI ::thThreadCleanup.Etat = threadInexistant ALORS
  ::thThreadCleanup = ThreadExécute(piCleanSessions, (), threadContexteGlobal)
FIN

PROCÉDURE INTERNE piCleanSessions()
BOUCLE
  ThreadPause(60s)

  SI ThreadArrêtDemandé() ALORS
    SORTIR
  FIN

```

```

        ::__sessionsCleanup()
    FIN
FIN

```

RENOYER *Vrai*

Méthode Create

```

PROCÉDURE Create(LOCAL pTimeout est entier = 3600) : booléen //Session Timeout session in seconds

duTimeout est une Durée

duTimeout..EnSecondes = pTimeout

RENOYER :Create(duTimeout)

```

Méthode Destroy

```

// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'Destroy' :
//
// Syntaxe :
// [ <Résultat> = ] Destroy ()
//
// Paramètres :
//   Aucun
// Valeur de retour :
//   multi-valeur : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
//   Détaillez ici le fonctionnement.
//   Précisez les cas particuliers et les limites.
//
// Exemple :
//   Indiquez ici un exemple d'utilisation.
//
FONCTION Destroy() : (booléen, entier)

cFileName est chaîne

cFileName = ::cSessionsFolder+:ID+".session"

SémaphoreDébut(::SEMAPHORE_SESSIONS+:ID)

SI fFichierExiste(cFileName) _ET_ PAS fSupprime(cFileName) ALORS
    SémaphoreFin(::SEMAPHORE_SESSIONS+:ID)
    :eLastErrorCode = ::ErrorSessionCannotDestroy
    RENOYER (Faux, :eLastErrorCode)
FIN

SémaphoreDétruit(::SEMAPHORE_SESSIONS+:ID)
RENOYER (Vrai, 0)

```

Destructeur

```

// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'Destructeur' :
//
// Syntaxe :
// Destructeur ()
//
// Paramètres :
//   Aucun

```

```
// Valeur de retour :
//  Aucune
//
// Notes :
//  Détaillez ici le fonctionnement.
//  Précisez les cas particuliers et les limites.
//
// Exemple :
//  Indiquez ici un exemple d'utilisation.
//
PROCÉDURE Destructeur()

SI ::thThreadCleanup..Etat <> threadInexistent ALORS
    :thThreadCleanup.DemandeArrêt()
FIN
```

Récupération de la propriété CipherringKey

FONCTION PUBLIQUE CipherringKey() : Buffer

RENOYER ::bCipherringKey

Affectation de la propriété CipherringKey

PROCÉDURE PUBLIQUE CipherringKey(LOCAL pKey est Buffer)

```
::bCipherringKey = pKey
::bKeyHash = HashChaîne(HA_HMAC_SHA_256, ::bCipherringKey)
```

Récupération de la propriété ID

FONCTION PUBLIQUE ID() : chaîne

RENOYER :cID

Affectation de la propriété ID

PROCÉDURE PUBLIQUE ID(LOCAL pID est chaîne)

```
:cID = pID
```

Récupération de la propriété IsCheckSignature

FONCTION PUBLIQUE IsCheckSignature() : booléen

RENOYER ::bCheckSignature

Affectation de la propriété IsCheckSignature

PROCÉDURE PUBLIQUE IsCheckSignature(LOCAL pCheck est booléen)

```
::bCheckSignature = pCheck
```

Récupération de la propriété LastErrorCode

FONCTION PUBLIQUE LastErrorCode() : entier

RENOYER :eLastErrorCode

Récupération de la propriété Salt

FONCTION PUBLIQUE Salt() : Buffer

RENOYER ::cSessionSalt

Affectation de la propriété Salt

PROCÉDURE PUBLIQUE Salt(pSaltValue est Buffer)

::cSessionSalt = pSaltValue

Récupération de la propriété SessionsDirectory

FONCTION PUBLIQUE SessionsDirectory() : chaîne

RENOYER ::cSessionsFolder

Affectation de la propriété SessionsDirectory

PROCÉDURE PUBLIQUE SessionsDirectory(LOCAL pDirectory est chaîne)

::cSessionsFolder = ComplèteRep(pDirectory)

Méthode GetAllData

FONCTION GetAllData()

```
jsSession    est JSON
bOK          est booléen
eErr         est entier
taData       est tableau associatif de Variant
```

SémaphoreDébut(::SEMAPHORE_SESSIONS+:ID)

```
(bOK, eErr) = :Check()
SI PAS bOK ALORS
    RENOYER (Faux, eErr)
FIN
```

```
(bOK, eErr, jsSession) = :__sessionRead()
SI PAS bOK ALORS
    RENOYER (Faux, eErr)
FIN
```

```
POUR TOUT vVal, vTag de jsSession.data
    taData[vTag] = vVal
FIN
```

RENOYER (Vrai, taData)

```
FIN:
    SémaphoreFin(::SEMAPHORE_SESSIONS+:ID)
```

Méthode GetData

```
// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'GetData' :
//
// Syntaxe :
//[ <Résultat> = ] GetData (<pTag> est chaîne)
//
// Paramètres :
//   pTag (chaîne ANSI) : <indiquez ici le rôle de pTag>
// Valeur de retour :
//   multi-valeur : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
```

```

// Détaillez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION GetData(LOCAL pTag est chaîne) //: (booleen, variant or error)

jsSession      est JSON
bOK             est booléen
eErr           est entier

SémaphoreDébut(::SEMAPHORE_SESSIONS+:ID)

(bOK, eErr) = :Check()
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

(bOK, eErr, jsSession) = :__sessionRead()
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

QUAND EXCEPTION DANS
    RENVOYER (Vrai, {"jsession.data."+pTag, indVariable})
FAIRE
    ExceptionActive()
    RENVOYER (Faux, ::ErrorSessionCannotGetValue)
FIN

FIN:
    SémaphoreFin(::SEMAPHORE_SESSIONS+:ID)

```

Méthode GetErrorMessage

```

// Résumé : <indiquez ici ce que fait la FONCTION>
// Description des paramètres d'entrée/sortie de 'GetErrorMessage' :
//
// Syntaxe :
//[ <Résultat> = ] GetErrorMessage (<pErrorMessage> est entier)
//
// Paramètres :
// pErrorMessage (entier) : <indiquez ici le rôle de pErrorMessage>
// Valeur de retour :
// chaîne ANSI : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
// Détaillez ici le fonctionnement.
// Précisez les cas particuliers et les limites.
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION GetErrorMessage(LOCAL pErrorMessage est entier) : chaîne

cRes est chaîne
SELON pErrorMessage
    CAS ::ErrorSessionExpired :          cRes = "Session expirée"
    CAS ::ErrorSessionIllegal :          cRes = "Session illégale (ID invalide)"
    CAS ::ErrorSessionCannotLoad :       cRes = "Impossible de charger les données de la Session"
    CAS ::ErrorSessionCannotSave :       cRes = "Impossible de sauver les données de la Session"
    CAS ::ErrorSessionCannotSetValue :   cRes = "Impossible de mémoriser la valeur dans la Session"
    CAS ::ErrorSessionCannotDestroy :    cRes = "Impossible de détruire la Session"
    CAS ::ErrorSessionCannotDecipher :   cRes = "Impossible de déchiffrer les données de la Session"

```

```

CAS ::ErrorSessionCannotGetValue : cRes =
  "Impossible de récupérer la valeur dans les données de la session"
AUTRES CAS :
  cRes = "Erreur inconnue"
FIN

RENOYER cRes

```

Méthode GetLastAccess

```

FONCTION GetLastAccess()

RENOYER = :GetData("_session.tsLastAccess")

```

Méthode GetTimestamp

```

FONCTION GetTimestamp()

RENOYER :GetData("_session.tsCreated")

```

Méthode Refresh

```

// Résumé : <indiquez ici ce que fait la procédure>
// Description des paramètres d'entrée/sortie de 'Refresh' :
//
// Syntaxe :
// [ <Résultat> = ] Refresh ()
//
// Paramètres :
//   Aucun
// Valeur de retour :
//   multi-valeur : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Notes :
//   Détaillez ici le fonctionnement.
//   Précisez les cas particuliers et les limites.
//
// Exemple :
//   Indiquez ici un exemple d'utilisation.
//
FONCTION Refresh() : (booléen, entier)

jsSession      est JSON
bOK             est booléen
eErr           est entier
sResVal        est chaîne

SémaphoreDébut(::SEMAPHORE_SESSIONS+:ID)

(bOK, eErr) = :Check()
SI PAS bOK ALORS
  RENOYER (Faux, eErr)
FIN

(bOK, eErr, jsSession) = :__sessionRead()
SI PAS bOK ALORS
  RENOYER (Faux, eErr)
FIN

jsSession._session.TsLastAccess  = :__TimeStamp()

(bOK, eErr, sResVal) = :__sessionWrite(jsSession)
SI PAS bOK ALORS

```

```

    RENVOYER (Faux, eErr)
FIN

RENVOYER (Vrai, 0)

FIN:
    SémaphoreFin(::SEMAPHORE_SESSIONS+:ID)

```

Méthode SetCipherringKey

```

//Conservé pour compatibilité avec les versions précédentes. Il est préférable d'utiliser la propriété
:CipherringKey

//Kept for compatibiliy with earlier versions. Using the :CipherringKey property is better.

PROCÉDURE SetCipherringKey(LOCAL pKey est Buffer)

:CipherringKey = pKey

```

Méthode SetData

```

//Fonction dépréciée, utiliser plutôt la syntaxe avec (pValues) en paramètre
FONCTION SetData(LOCAL pTag est chaîne, pValue est Variant) : (booléen, entier)

jsSession      est JSON
vSessionData est Variant
bOK             est booléen
eErr            est entier
sResVal        est chaîne

SémaphoreDébut(::SEMAPHORE_SESSIONS+:ID)

(bOK, eErr) = :Check()
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

(bOK, eErr, jsSession) = :__sessionRead()
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

jsSession._session.TsLastAccess = :__TimeStamp()
vSessionData = JSONVersVariant(jsSession)

QUAND EXCEPTION DANS
    {"vSessionData.data."+pTag, indVariable} = pValue
FAIRE
    ExceptionActive()
    RENVOYER (Faux, ::ErrorSessionCannotSetValue)
FIN

jsSession = VariantVersJSON(vSessionData)

(bOK, eErr, sResVal) = :__sessionWrite(jsSession)
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

RENVOYER (Vrai, 0)

FIN:
    SémaphoreFin(::SEMAPHORE_SESSIONS+:ID)

```

Méthode SetData

```

FONCTION SetData(pValues est Variant) : (booléen, entier)

jsSession      est JSON
vSessionData est Variant
bOK             est booléen
eErr           est entier
sResVal        est chaîne

SémaphoreDébut(::SEMAPHORE_SESSIONS+:ID)

(bOK, eErr) = :Check()
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

(bOK, eErr, jsSession) = :__sessionRead()
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

jsSession._session.TsLastAccess = :__TimeStamp()
vSessionData = JSONVersVariant(jsSession)

POUR TOUT vValue, vTag de pValues
    QUAND EXCEPTION DANS
        {"vSessionData.data."+vTag, indVariable} = vValue
    FAIRE
        ExceptionActive()
        RENVOYER (Faux, ::ErrorSessionCannotSetValue)
    FIN
FIN

jsSession = VariantVersJSON(vSessionData)

(bOK, eErr, sResVal) = :__sessionWrite(jsSession)
SI PAS bOK ALORS
    RENVOYER (Faux, eErr)
FIN

RENOYER (Vrai, 0)

FIN:
    SémaphoreFin(::SEMAPHORE_SESSIONS+:ID)

```

IrRoute

Code

Déclaration de IrRoute

```

<BLOC const>
  CONSTANTE
    //Methods
    MethodGET      = "GET"
    MethodPOST     = "POST"
    MethodPUT      = "PUT"
    MethodDELETE   = "DELETE"
    MethodHEAD     = "HEAD"
    MethodPATCH   = "PATCH"
    MethodOPTIONS  = "OPTIONS"

    SEMAPHORE_CUSTOM_ROUTES = "SEM_CUSTOM_ROUTES_"
  FIN
<FIN>

IrRoute est une Classe
  Route      est chaîne
  Method     est chaîne
  ProcFonction est procédure
  Timeout    est Durée
  UseGlobalOptionHeaders est booléen
  NoRegex    est booléen

  OptionsHeaders est tableau associatif (ccSansCasse+AvecDoublon) de chaînes

  PRIVÉ
    UUID      est UUID
    vCustomData est Variant
FIN

```

Constructeur

```

PROCÉDURE Constructeur()

vEmpty est Variant

:UUID = DonneUUID()
SémaphoreCrée(::SEMAPHORE_CUSTOM_ROUTES+:UUID, partageAucun)

:CustomData = vEmpty

```

Destructeur

```

PROCÉDURE Destructeur()

SémaphoreDétruit(::SEMAPHORE_CUSTOM_ROUTES+:UUID)

```

Récupération de la propriété CustomData

```
PROCÉDURE PUBLIQUE CustomData()
```

```
vCustomData est Variant
```

```
SémaphoreDébut(::SEMAPHORE_CUSTOM_ROUTES+:UUID)
```

```
vCustomData = :vCustomData
```

```
SémaphoreFin(::SEMAPHORE_CUSTOM_ROUTES+:UUID)
```

```
RENVOYER vCustomData
```

Affectation de la propriété CustomData

```
PROCÉDURE PUBLIQUE CustomData(pCustomData est Variant)
```

```
SémaphoreDébut(::SEMAPHORE_CUSTOM_ROUTES+:UUID)
```

```
:vCustomData = pCustomData
```

```
SémaphoreFin(::SEMAPHORE_CUSTOM_ROUTES+:UUID)
```

Partie 3

Table des matières

Projet lightREST3

Partie 1	Projet	3
	Code	3
	Statistiques sur le code	3

Partie 2	Classe	5
	IrResponse	5
	Code	5
	IrRequest	12
	Code	12
	IrServer	17
	Code	17
	IrSession	35
	Code	35
	IrRoute	46
	Code	46